

Mikrokomputery

Jan Bielecki

JĘZYK FORTH

FORTH

FORTH

FORTH

FORTH

Język FORTH

Moim przyjaciołom
Joli i Pambosowi Kalopsidiotis

Mikrokomputery

Komitet Redakcyjny

Jan Bielecki

Język FORTH



Wydawnictwa Naukowo-Techniczne
Warszawa 1988

Opiniodawca *Jędrzej Wróblewski*
Redaktor *Ewa Zdanowicz*
Redaktor techniczny *Bogumił Marczak*
Projektant serii *Juliusz Rybicki*
Okładkę i strony tytułowe projektował *Andrzej Pilich*

681.3.06

W książce opisano język programowania Forth. Podano pojęcia podstawowe dotyczące tego języka, przedstawiono wybrane operacje stosowe i słownikowe oraz omówiono zasady definiowania operatorów, kompilatorów i instrukcji. Przedstawiono możliwości rozbudowywania języka, współpracę z urządzeniami zewnętrznymi i oprogramowanie grafiki.
Książka jest przeznaczona dla czytelników interesujących się programowaniem mikrokomputerów.

ISBN 83-204-0930-6

Spis treści

Wstęp

Język programowania Forth jest zaliczany do „asemblerowych” języków wysokiego poziomu. Umożliwia on programowanie strukturalne zbliżone do programowania w języku maszynowym i z tego względu jest często wykorzystywany jako narzędzie do programowania systemowego. Jedną z głównych zalet języka Forth jest jego rozbudowywalność, przejawiająca się możliwością definiowania w nim nowych operatorów i instrukcji, w tym operatorów ułatwiających posługiwanie się rozkazami maszynowymi.

Forth jest językiem interakcyjnym, co oznacza, że polecenia wykonania operacji — pochodzące z klawiatury terminala — są natychmiast realizowane. Odbywa się to bez charakterystycznego dla większości innych języków, czasochłonnego procesu kompilacji i konsolidacji.

Forth występuje w kilku dialektach, z których najbardziej rozpowszechniony jest fig-Forth. Poza nim na uwagę zasługują poly-Forth, Forth-79 i Forth-83. Wszystkie te dialekty łączy wspólna koncepcja i biegłe opanowanie jednego z nich wystarcza do samodzielnego posłużenia się dowolnym z pozostałych.

Niniejsze opracowanie jest poświęcone prezentacji dialektu fig-Forth. W kolejnych rozdziałach przedstawiono strukturę procesora języka Forth, zasady definiowania operatorów i instrukcji, tworzenia podsłowników i konstruowania programów hybrydowych. W celu ułatwienia posługiwania się tekstem w części II zamieszczono opisy operatorów w porządku alfabetycznym — wyznaczonym przez kod ASCII — a na końcu pracy ich indeks.

Należy oczekiwać, że częste odwoływanie się do tych opisów powinno znacznie ułatwić studiowanie tekstu.

Część I

Opis języka

1. Pojęcia podstawowe

Program napisany w języku Forth składa się z *sekwencji słów. Słowa* składają się ze *spójnych ciągów znaków* różnych od spacji, a ogranicznikiem słowa jest spacja, znak o kodzie zero albo powrót karetki. Grupy słów są najczęściej łączone w *wiersze*, a wiersze umieszczane w pamięci zewnętrznej w postaci tzw. *ekranów*. Każdy ekran składa się z 1024 znaków zgrupowanych w 16 wierszy po 64 znaki. Wiersze są numerowane od 0 do 15.

W repertuarze słów języka Forth znajduje się słowo, którego zinterpretowanie umożliwia wykonanie programu zapamiętanego w ekranie. Ponieważ Forth jest językiem interakcyjnym, możliwe jest również interpretowanie słów wprowadzanych na bieżąco z klawiatury terminala. Należą do nich m.in. słowa umożliwiające tworzenie i modyfikowanie zawartości ekranów.

Zinterpretowanie słowa powoduje wykonanie określonej operacji. Odbywa się to pod nadzorem *procesora* języka Forth, tj. tego czynnika, który nadzoruje interpretację oraz zarządza zasobami niezbędnymi do jej wykonania. Do zasobów tych należą przede wszystkim: *słownik operatorów* — — zawierający definicje wszystkich dostępnych operacji oraz dwa stosy: *stos parametrów* — na którym są umieszczane argumenty i rezultaty operacji oraz *stos powrotów* — wykorzystywany głównie przez interpreter, ale również nadający się do przechowywania danych.

Ciąg słów przekazywanych do interpretacji jest nazywany *strumieniem wejściowym.* Źródłem strumienia wejściowego może być ekran pamięci zewnętrznej albo klawiatura terminala. Każde słowo strumienia wejściowego jest poddawane natychmiastowej interpretacji. Przebieg interpretacji, nadzorowany

przez procesor języka Forth, zależy od chwilowego stanu interpretera. Interpreter może znajdować się w *stanie wykonywania* albo w *stanie definiowania*. Bezpośrednio po zainicjowaniu procesora, interpreter znajduje się w stanie wykonywania, a strumień wejściowy jest związany z klawiaturą terminala.

Niezależnie od stanu interpretera, zinterpretowanie najbliższego słowa znajdującego się w strumieniu wejściowym polega na wyszukaniu w słowniku operatorów, definicji operacji identyfikowanej przez dane słowo. Jeśli w słowniku nie zostanie znaleziona poszukiwana definicja, to wymaga się, aby interpretowane słowo było liczbą. W przeciwnym razie jest sygnalizowany błąd, rozpatrywane słowo jest ignorowane, a strumień wejściowy zostaje związany z klawiaturą terminala.

Jeśli interpreter znajduje się w stanie wykonywania, a interpretowane słowo identyfikuje operator, to jest wykonywana operacja związana z tym operatorem. Jeśli natomiast interpretowane słowo indentyfikuje liczbę, to jest wykonywana operacja polegająca na umieszczeniu na stosie parametrów procesora danej liczbowej, która ma wartość rozpatrywanej liczby.

Jeśli interpreter znajduje się w stanie definiowania, to sposób interpretowania słów zależy od tego, czy identyfikują one operatory czynne czy bierne. *Operatory czynne* są związane z operacjami czynnymi — realizowanymi w obu stanach interpretera, natomiast *operatory bierne* są związane z operacjami biernymi — realizowanymi jedynie wtedy, kiedy interpreter znajduje się w stanie wykonywania.

Jeśli interpreter znajduje się w stanie definiowania, a interpretowane słowo identyfikuje operator czynny, to jest wykonywana operacja związana z tym operatorem. Jeśli interpretowane słowo identyfikuje operator bierny, to w słowniku operatorów zostaje umieszczony adres pola kodu tego operatora. Adres ten będzie nazywany *wskazaniem* operatora*). Jeśli interpretowane słowo identyfikuje liczbę, to w słowniku operatorów jest umieszczone wskazanie operatora LIT, a za nim dana słowowa mająca wartość wspomnianej liczby. W przypadku liczby podwojonej dokładności, zajmującej w pamięci operacyjnej dwa słowa, umieszcza się w słowniku wskazanie operatora LIT, za nim słowo zawierające mniej znaczącą część liczby, ponownie wskazanie operatora LIT, a za nim słowo zawierające bardziej znaczącą część liczby.

Ze względu na to, że zinterpretowanie słowa, które nie jest operatorem ale ma postać liczby, powoduje wykonanie ustalonej, opisanej wyżej operacji, dogodnie jest uznawać takie słowa za niejawnie zdefiniowane operatory bierne związane z opisanymi wyżej operacjami. Z tego względu nic nie stoi na przeszkodzie, aby rozpatrywane liczby także nazywać operatorami. W tym sensie można np. mówić o wykonaniu operacji 20.

---

*) Zgodnie z przyjętą definicją, zwrot "umieszczenie wskazania operatora" jest skrótem od "umieszczenie adresu pola kodu operatora".

Poprzestając na tym omówieniu zasad interpretowania słów strumienia wejściowego pozostaje zwrócić uwagę na to, iż niekiedy zinterpretowanie słowa powoduje wprowadzenie i wykorzystanie kolejnej porcji znaków strumienia bez skierowywania jej do interpretacji. Taki sposób postępowania umożliwia przekazywanie argumentów operatorów bezpośrednio w samym strumieniu wejściowym.

Przykłady

a. Interpretowanie strumienia wejściowego pochodzącego z klawiatury terminala

```
( PRINT )  : PRINT
             DUP  ." TOP = "  .  ;
20  PRINT
```

Jeśli trzy przytoczone wiersze stanowią strumień wejściowy pochodzący z terminala, to przebieg interpretacji jest następujący:

- Napis (zostanie uznany za słowo identyfikujące operator czynny ((*left-paren*). Argumentem tego operatora jest ciąg znaków strumienia wejściowego aż do najbliższego znaku) włącznie. Ponieważ wykonanie operacji (sprowadza się do zignorowania tego ciągu znaków, konstrukcję (PRINT) można uznać za komentarz. Z tego powodu pierwszy napis PRINT nie jest uznawany za słowo i nie podlega interpretacji.
- Napis : zostanie uznany za słowo identyfikujące operator czynny : (*colon*). Argumentem tego operatora, wykorzystywanego do definiowania nowych operatorów, jest najbliższe słowo strumienia wejściowego, w danym przypadku słowo PRINT. Wykonanie operacji : spowoduje umieszczenie w słowniku nagłówka definicji operatora PRINT i ustawienie interpretera w stan definiowania.
- Napis DUP zostanie uznany za słowo identyfikujące operator bierny DUP. Ponieważ interpreter znajduje się w stanie definiowania, zinterpretowanie słowa DUP spowoduje umieszczenie w słowniku wskazania definicji operatora DUP. Wskazanie to zostanie umieszczone bezpośrednio za nagłówkiem definicji operatora PRINT.
- Napis ." zostanie uznany za słowo identyfikujące operator czynny ." (*dot-quote*). Mimo iż interpreter znajduje się w stanie definiowania, nastąpi wykonanie operacji .". Stanie się tak, ponieważ ." jest operatorem czynnym. Ponieważ argumentem operatora ." jest ciąg znaków strumienia wejściowego aż do znaku " włącznie, najbliższym interpretowanym słowem nie będzie TOP lecz . (kropka). W następstwie wykonania operacji ." w słowniku operatorów zostanie umieszczone wskazanie pomocniczego operatora (."), a za nim tekst

poprzedzony licznikiem znaków, składający się z 6 znaków występujących między spacją kończącą napis ." a najbliższym znakiem ". Podobnie jak wskazanie powstałe ze zinterpretowania słowa DUP, rozszerzą one definicję operatora PRINT.

• Napis . zostanie uznany za słowo identyfikujące operator bierny . (*dot*). Spowoduje to umieszczenie w słowniku wskazania definicji tego operatora.

• Napis ; zostanie uznany za słowo identyfikujące operator czynny ; (*semicolon*). Spowoduje to wykonanie operacji ; mimo iż interpreter znajduje się w stanie definiowania. Rezultatem wykonania tej operacji będzie umieszczenie w słowniku wskazania operatora pomocniczego ;S i zakończenie definiowania operatora PRINT. Ponadto interpreter zostanie ustawiony w stan wykonywania.

• Napis 20 zostanie uznany za słowo identyfikujące liczbę. Spowoduje to umieszczenie na stosie parametrów danej słowowej o wartości 20.

• Napis PRINT zostanie uznany za słowo identyfikujące operator PRINT. Ponieważ definicja operatora PRINT została właśnie utworzona, użycie słowa PRINT w strumieniu wejściowym jest poprawne i powoduje wykonanie operacji PRINT. Jak wynika z przytoczonego opisu, definicja ta składa się z nagłówka, wskazania operatora DUP, wskazania operatora (."), za którym następuje tekst TOP = , wskazania operatora . oraz wskazania operatora ;S. Zinterpretowanie takiej definicji spowoduje: powielenie — na stosie parametrów — danej o wartości 20 (operacja DUP), wyprowadzenie tekstu TOP = zawartego w definicji (operacja (.")), wyprowadzenie danej znajdującej się na stosie parametrów (operacja .) i powrót do dalszego interpretowania strumienia wejściowego (operacja ;S). Łącznie spowoduje to wyprowadzenie napisu

TOP = 20

zakończonego parą liter OK (*all-right*). Napis ten wyraża gotowość procesora do kontynuowania interpretacji.

b. Interpretowanie strumienia wejściowego pochodzącego z pamięci zewnętrznej.

Jeśli przyjąć, że dwa pierwsze wiersze strumienia wejściowego z poprzedniego przykładu zostały umieszczone w ekranie o numerze 127, to związanie strumienia wejściowego z klawiaturą terminala i wprowadzenie z niej wiersza

127 LOAD 30 PRINT

wywoła następujący przebieg interpretacji:

• Napis 127 zostanie uznany za słowo identyfikujące operację 127. Zinterpretowanie tej operacji spowoduje umieszczenie — na stosie parametrów — danej o wartości 127.

• Napis LOAD zostanie uznany za słowo identyfikujące operator LOAD. Zinterpretowanie operacji LOAD spowoduje zdjęcie ze stosu parame-

trów znajdującej się tam danej i potraktowania jej wartości jako numeru ekranu. Następnie dokona się przełączenie strumienia wejściowego z klawiatury na ekran i zinterpretowanie treści ekranu, w tym przypadku o numerze 127. Po zakończeniu interpretowania ekranu w słowniku pojawi się definicja operatora PRINT, po czym nastąpi przełączenie źródła strumienia wejściowego tak, aby była nim klawiatura.

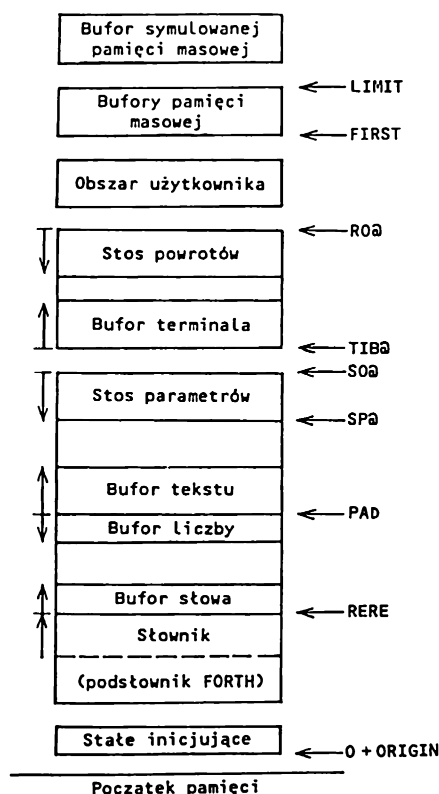
o Napis 30 zostanie uznany za słowo identyfikujące operator 30. Wykonanie tej operacji spowoduje umieszczenie — na stosie parametrów — danej o wartości 30.

o Napis PRINT zostanie uznany za słowo identyfikujące operację PRINT. Wykonanie operacji PRINT spowoduje wyprowadzenie napisu

TOP = 30

oraz pary liter OK jak w poprzednim przykładzie. □

Najważniejszym elementem procesora języka Forth jest *słownik operatorów*, nazywany krótko *słownikiem*. To w nim właśnie są zapisane nazwy



1.1. Organizacja pamięci

poszczególnych operatorów oraz określone są czynności wykonywane podczas interpretowania poszczególnych słów strumienia wejściowego. Słownik ma *strukturę drzewiastą* i składa się z szeregu *podsłowników*. Bezpośrednio po zainicjowaniu interpretacji składa się on z ustalonego zestawu definicji, ujętego w podsłownik o nazwie FORTH. W miarę postępowania interpretacji podsłownik ten może być rozbudowywany przez wprowadzenie nowych definicji. Możliwe jest także definiowanie nowych podsłowników. Ta rozbudowywalność jest jedną z cech wyróżniających Forth wśród innych języków programowania. Umożliwia ona wzbogacenie podstawowej wersji języka o nowe operatory i instrukcje. Stanowi znaczne ułatwienie tworzenia wyspecjalizowanych zastosowań, zwłaszcza o charakterze systemowym. Nie bez znaczenia jest także fakt, że tworzenie wykonywalnego programu w języku Forth odbywa się bez konsolidacji. Efektem interakcyjnego wykonania operacji jest bowiem utworzenie danej — najczęściej umieszczanej na jednym ze stosów albo skompilowanie nowej definicji operatora, tj. umieszczenie jej w słowniku.

Poza stosami i słownikiem, procesor języka Forth zarządza szeregiem innych obszarów pamięci operacyjnej, przedstawionych na rys. 1.1. Należą do nich: *obszar stałych inicjujących* — wykorzystywanych w chwili rozpoczęcia pracy procesora, *słownik* — zawierający definicje operatorów, *bufor słowa* — zawierający kolejno interpretowane słowa strumienia wejściowego, *bufor liczby* — w którym odbywa się kompletowanie cyfr wyprowadzanej danej liczbowej, *bufor tekstu* — najczęściej wykorzystywany przez edytor, *bufor terminala* — w którym są umieszczane wiersze strumienia wejściowego, *bufory współpracy z pamięcią zewnętrzną* i ewentualnie — w okrojonych implementacjach — *bufor symulowanej pamięci masowej*. Adresy poszczególnych obszarów można uzyskać na stosie parametrów, interpretując słowa lub sekwencje słów podane z prawej strony rysunku.

2. Stos parametrów

Wymieniony uprzednio *stos parametrów* jest obszarem pamięci operacyjnej wykorzystywanym do przechowywania argumentów i rezultatów operacji. Stos ten jest strukturą typu LIFO (*last in — first out*), co oznacza, że dane są zawsze umieszczane na szczycie stosu i tylko ze szczytu są zdejmowane. Stos parametrów może pomieścić ograniczoną liczbę danych słowowych, którym można nadawać różne interpretacje. Można np. traktować je jak liczby całkowite ze znakiem i bez znaku, pary znaków w kodzie ASCII albo dane logiczne. Jeśli zaistnieje potrzeba umieszczenia na stosie parametrów danej dwusłowowej, to rozbija się ją na dwie dane słowowe i umieszcza na stosie w takiej kolejności, aby bardziej znaczące słowo danej znajdowało się bliżej wierzchołka stosu.

W miarę umieszczania na stosie i zdejmowania z niego danych słowowych, wierzchołek stosu ulega przesunięciu. Chociaż rzadko jest to potrzebne, można określić jego położenie posługując się operatorem SP@. Udostępnia on na stosie parametrów adres ostatniego bajtu pamięci operacyjnej zajętego przez stos parametrów tuż przed wykonaniem tej operacji. Należy nadmienić, że w większości implementacji stos parametrów rozrasta się w kierunku niższych adresów pamięci, a więc umieszczenie na stosie nowej danej powoduje zmniejszenie wartości adresu określającego wierzchołek stosu.

Jak już wyjaśniano, zinterpretowanie słowa, któremu nie odpowiada definicja słownikowa, a które ma postać liczby, powoduje umieszczenie — na stosie parametrów — danej słowowej o wartości tej liczby. W szczególności zinterpretowanie pary słów

```
20 -50
```

spowoduje umieszczenie — na stosie parametrów — danej o wartości 20, a następnie danej o wartości −50 (w zapisie uzupełnieniowym do 2).

Zinterpretowanie w takiej sytuacji pary słów

+ .

spowoduje kolejno: zastąpienie pary danych na szczycie stosu daną o wartości ich sumy (operacja +) oraz zdjęcie tej danej ze stosu i wyprowadzenie (operacja .). Tym samym rezultatem zinterpretowania ciągu słów

20 −50 + .

będzie wyprowadzenie liczby −30 zakończonej spacją i przywrócenie stanu stosu parametrów jak przed wykonaniem tej interpretacji.

Wykonywanie bardziej złożonych działań na stosie parametrów wymaga znajomości sposobu odwzorowywania wyrażeń nawiasowych na wyrażenia w *odwrotnej notacji polskiej.* Zgodnie z tą notacją działanie takie jak np. 2 * (3+4) przybiera postać 2 3 4 + *, co oznacza, że na stosie zostają umieszczone kolejno dane 2, 3 i 4, następnie wykonana operacja +, a po niej operacja *. Wykonanie operacji + powoduje zdjęcie ze stosu danych 4 i 3, wykonanie dodawania i umieszczenie na stosie rezultatu 7. Wykonanie operacji * powoduje zdjęcie ze stosu danych 7 i 2, wykonanie mnożenia i umieszczenie na stosie rezultatu 14. Tym samym, nie posługując się nawiasami, można uzyskać ten sam rezultat. Kilka innych przykładów przekształcenia wyrażeń algebraicznych w ich równoważniki w notacji odwrotnej przedstawiono na rys. 2.1.

Wyrażenie algebraiczne	Wyrażenie w notacji odwrotnej
2 + 5	2 5 +
3 * (2 + 5)	3 2 5 + *
(2 + 5) * 3	2 5 + 3 *
2 - 3 * (4 + 5)	2 3 4 5 + * -
(2 + 3) / (4 - 5)	2 3 + 4 5 - /
2 + (3 - 4 * 5)	2 3 4 5 * - +

2.1. Wyrażenia algebraiczne w notacji odwrotnej

Umieszczenie na stosie parametrów, danych podwojonej dokładności, wymaga zastosowania zmodyfikowanego zapisu liczb. Słowa wyrażające takie liczby powinny zawierać co najmniej jeden znak . (kropka), umieszczony przed wszystkimi cyframi liczby, w ich obrębie albo za wszystkimi cyframi. W szczególności, zinterpretowanie pary słów

20. −5.0

spowoduje umieszczenie — na stosie parametrów — danej dwusłowowej o wartości 20, a następnie danej dwusłowowej o wartości −50. Zsumowanie takich danych i wyprowadzenie rezultatu wymaga użycia operatorów przystosowanych do wykonywania operacji na danych podwojonej dokładności,

a mianowicie: operatora D+ (dla dodawania) i D. (dla wyprowadzania). Oznacza to, że wykonanie przykładowej sekwencji

20. −5.0 D+ D.

powoduje zsumowanie umieszczonych na stosie danych dwusłowowych i wyprowadzenie rezultatu −30.

Gdyby zamiast przytoczonej sekwencji posłużono się sekwencją

20. −5.0 + .

(z operacjami dla danych pojedynczej dokładności), to operacja + dotyczyłaby jedynie dwóch jednosłowowych części danej dwusłowowej o wartości −50. Ponieważ dana dwusłowowa o tej wartości jest reprezentowana na stosie jako para danych słowowych o wartościach −50 i −1, rezultatem operacji byłoby wyprowadzenie liczby −51. Na stosie pozostałyby jeszcze dane słowowe o wartościach 20 i 0. Wynika to z faktu, że procesor nie rozpoznaje typu danych znajdujących się na stosie, a sposób operowania tymi danymi leży całkowicie w gestii programującego.

Przykład

Zinterpretowanie sekwencji słów

20. 30 0 D+ . .

spowoduje wykonanie następujących czynności:

- Umieszczenie — na stosie parametrów — danej dwusłowowej o wartości 20.
- Umieszczenie — na stosie parametrów — danej słowowej o wartości 30.
- Umieszczenie — na stosie parametrów — danej słowowej o wartości 0.
- Potraktowanie danych słowowych o wartości 30 i 0 jak danej dwusłowowej o wartości 30 i zsumowanie jej z daną dwusłowową o wartości 20.
- Potraktowanie bardziej znaczącej części danej dwusłowowej o wartości 50 jako danej słowowej i wyprowadzenie jej w postaci liczby 0.
- Potraktowanie mniej znaczącej części danej dwusłowowej jako danej słowowej i wyprowadzenie liczby 50. □

3. Operacje stosowe i słownikowe

Większość operacji języka Forth dotyczy stosu parametrów. Z tego względu w opisach operacji jest stosowana konwencja przytaczania stanu szczytowych elementów tego stosu przed i po wykonaniu operacji. Zgodnie z tą konwencją, zapis taki jak np.

(*a b* -- *c*)

oznacza, że argumentami operacji są dwa szczytowe elementy stosu parametrów: *a* i *b*, które po wykonaniu operacji zostaną zastąpione elementem *c*. O tym czy spowoduje to zwiększenie, czy zmniejszenie rozmiaru stosu parametrów, decyduje charakter danych. Jeśli np. dana *a* jest dwusłowowa, a dane *b* i *c* są słowowe, to stos parametrów skróci się o jedno słowo.

Dysponując przytoczonym zapisem argumentów i rezultatów operacji, określającym zmiany dokonujące się na stosie parametrów, można przystąpić do krótkiego przeglądu ważniejszych operacji języka Forth. Na wstępie zostaną omówione podstawowe operacje wejścia/wyjścia i operacje stosowe, ponieważ ułatwią one konstruowanie przykładów. Następnie zostaną przedstawione operacje arytmetyczne, pamięciowe, relacje, operacje logiczne i operacje słownikowe. Pełen wykaz operacji zamieszczono w części II.

Operacje wejścia/wyjścia

Operacje wejścia/wyjścia służą do wprowadzania i wyprowadzania liczb i znaków oraz do sprawdzania stanu klawiatury terminala. Podstawowymi operacjami wejścia/wyjścia są:
. (wyprowadź-daną-słowową-jako-liczbę), D. (wyprowadź-daną-dwusłowową-jako-liczbę), .R (wprowadź-daną-słowową-jako-liczbę-wyrównaną-prawostronnie), D.R (wyprowadź-daną-dwusłowową-jako-liczbę-wyrówna-

ną-prawostronnie), EMIT (wyprowadź-znak), CR (przejdź-do-nowego-wiersza), KEY (wprowadź-znak), ?TERMINAL (zbadaj-bufor-terminala).

. (*a* --)

Wykonanie operacji . powoduje zdjęcie — ze stosu parametrów — danej słowowej *a* i wyprowadzenie jej jako liczby zakończonej jedną spacją. Jeśli wyprowadzana dana jest ujemna, to wyprowadzana liczba zaczyna się od znaku — (minus).

D. (*a* --)

Wykonanie operacji D. powoduje zdjęcie — ze stosu parametrów — danej dwusłowowej *a* i wyprowadzenie jej jako liczby zakończonej jedną spacją. Jeśli wprowadzana dana jest ujemna, to wyprowadzana liczba zaczyna się od znaku — (minus).

.R (*a n* --)

Wykonanie operacji .R powoduje wyprowadzenie danej słowowej *a*, jako liczby wyrównanej prawostronnie, w polu wyjściowym o szerokości *n*.

D.R (*a n* --)

Wykonanie operacji D.R powoduje wyprowadzenie danej dwusłowowej *a*, jako liczby wyrównanej prawostronnie, w polu wyjściowym o szerokości *n*.

EMIT (*a* --)

Wykonanie operacji EMIT powoduje zdjęcie — ze stosu parametrów — danej słowowej *a* i wyprowadzenie znaku w kodzie ASCII, reprezentowanego w dolnym bajcie tej danej.

CR (--)

Wykonanie operacji CR powoduje wyprowadzenie znaków powodujących przejście do nowego wiersza.

?TERMINAL

(-- *t*)

(-- *f*)

Wykonanie operacji ?TERMINAL powoduje rozpatrzenie stanu buforu klawiatury terminala. Jeśli bufor ten zawiera jeszcze nie wprowadzony znak, to na stosie parametrów zostanie umieszczona dana *t* o wartości 1. W przeciwnym razie zostanie tam umieszczona dana *f* o wartości 0.

Przykłady

a. Wyprowadzenie bardziej znaczącej i mniej znaczącej części danej dwusłowowej znajdującej się na szczycie stosu parametrów

b. Wyprowadzenie bardziej znaczącej i mniej znaczącej części danej dwusłowowej w odrębnych wierszach

. CR .

c. Wyprowadzenie danej dwusłowowej znajdującej się na szczycie stosu parametrów, jako liczby umieszczonej prawostronnie w polu o szerokości 9 znaków

9 D.R □

Operacje stosowe

Operacje stosowe służą do wykonywania czynności dotyczących tych elementów stosu parametrów i stosu powrotów, które znajdują się w bezpośredniej bliskości ich szczytów. Do tej grupy operacji należą: DROP (usuń-ze-stosu), DUP (powiel-na-stosie), SWAP (zamień-miejscami), OVER (powiel-poprzedni), ROT (dokonaj-obrotu), >R (przenieś-na-stos-powrotów), R> (zdejmij-ze-stosu-powrotów), R (skopiuj-ze-stosu-powrotów).

DROP (*a* --)

Wykonanie operacji DROP powoduje usunięcie — ze stosu parametrów — danej słowowej *a*.

DUP (*a* -- *a a*)

Wykonanie operacji DUP powoduje powielenie — na stosie parametrów — danej słowowej *a*.

SWAP (*a b* -- *b a*)

Wykonanie operacji SWAP powoduje zamianę miejscami — na stosie parametrów — danych słowowych *a* i *b*.

OVER (*a b* -- *a b a*)

Wykonanie operacji OVER powoduje powielenie — na stosie parametrów — danej słowowej *a*.

ROT (*a b c* -- *b c a*)

Wykonanie operacji ROT powoduje przemieszczenie — na stosie parametrów — danych słowowych *a*, *b* i *c* w taki sposób, aby przyjęły kolejność *b*, *c* i *a*

>R (*a* --)

Wykonanie operacji >R powoduje przeniesienie danej słowowej *a* ze stosu parametrów na stos powrotów.

R> (-- *a*)

Wykonanie operacji R> powoduje przeniesienie danej słowowej *a* ze stosu powrotów na stos parametrów.

R (-- *a*)

Wykonanie operacji R powoduje skopiowanie danej słowowej *a* ze stosu powrotów na stos parametrów.

Przykłady

a. Umieszczenie dwóch danych na stosie, zamiana miejscami i wyprowadzenie w takiej kolejności, w jakiej były na nim umieszczane

30 40 SWAP . .

b. Umieszczenie dwóch danych na stosie i wyprowadzenie bez usuwania ich ze stosu

30 DUP . 40 DUP .

c. Umieszczenie trzech danych na stosie i wyprowadzenie w takiej kolejności, w jakiej były na nim umieszczane

30 40 50 SWAP ROT . . .

d. Wyprowadzenie sumy i różnicy dwóch danych

30 40 OVER OVER + . − . □

Odrębną grupę operacji stosowych stanowią te, które umożliwiają wykonywanie działań dotyczących stosu jako całości. Ponieważ nieumiejętne ich użycie może spowodować załamanie się interpretacji, należy stosować je ze szczególną ostrożnością. Do tej grupy operacji należą: SP! (wyzeruj-stos--parametrów), RP! (wyzeruj-stos-powrotów), Sp@ (określ-adres-szczytu-stosu--parametrów), RP@ (określ-adres-szczytu-stosu-powrotów), S0 (określ-adres--szczytu-pustego-stosu-parametrów), R0 (określ-adres-szczytu-pustego-stosu--powrotów).

SP! (cały-stos — —)

Wykonanie operacji SP! powoduje usunięcie ze stosu parametrów wszystkich znajdujących się na nim danych.

RP! (— —)

Wykonanie operacji RP! powoduje usunięcie ze stosu powrotów wszystkich znajdujących się na nim danych.

SP@ (— — *a*)

Wykonanie operacji SP@ powoduje umieszczenie — na stosie parametrów — adresu szczytu stosu parametrów tuż przed wykonaniem operacji SP@.

RP@ (— — *a*)

Wykonanie operacji RP@ powoduje umieszczenie — na stosie parametrów — adresu szczytu stosu powrotów tuż przed wykonaniem operacji RP@.

S0 (— — *a*)

Wykonanie operacji S0 powoduje umieszczenie — na stosie parametrów — adresu danej słowowej, której wartością jest adres szczytu pustego stosu parametrów.

R0 (— — *a*)

Wykonanie operacji R0 powoduje umieszczenie — na stosie para-

metrów — adresu danej słowowej, której wartością jest adres szczytu pustego stosu parametrów.

Przykład

Wyprowadzenie liczby danych słowowych znajdujących się na stosie parametrów.

SP@ S0 @ SWAP − 2 / .

• Wykonanie operacji @ powoduje potraktowanie danej słowowej, znajdującej się na szczycie stosu parametrów, jako adresu słowa i zastąpienie tego adresu tym słowem. □

Operacje arytmetyczne słowowe

Operacje arytmetyczne słowowe służą do wykonywania działań arytmetycznych na danych słowowych znajdujących się na stosie parametrów. Podstawowymi operacjami arytmetycznymi są + (plus), − (minus), * (gwiazdka), / (kreska-ukośna), 1+ (jeden-plus), 2+ (dwa-plus), MINUS (zmień-znak), ABS (wyznacz-wartość-bezwzględną), MOD (wyznacz-resztę-z-dzielenia). Wszystkie przytoczone tu operacje dotyczą danych słowowych.

+ (*a b* −− *c*)
Wykonanie operacji + powoduje zastąpienie — na stosie parametrów — danych słowowych *a* i *b* ich słowową sumą $c = a + b$.

− (*a b* −− *c*)
Wykonanie operacji − powoduje zastąpienie — na stosie parametrów — danych słowowych *a* i *b* ich słowową różnicą $c = a - b$.

* (*a b* −− *c*)
Wykonanie operacji * powoduje zastąpienie — na stosie parametrów — danych słowowych *a* i *b* ich słowowym iloczynem $c = a * b$.

/ (*a b* −− *c*)
Wykonanie operacji / powoduje zastąpienie — na stosie parametrów — danych słowowych *a* i *b* ich słowowym ilorazem $c = a/b$.

1+ (*a* −− *b*)
Wykonanie operacji 1+ powoduje zastąpienie — na stosie parametrów — danej słowowej *a* taką daną słowową *b*, że $b = a + 1$.

2+ (*a* −− *b*)
Wykonanie operacji 2+ powoduje zastąpienie — na stosie parametrów — danej słowowej *a* taką daną słowową *b*, że $b = a + 2$.

MINUS (a -- b)
Wykonanie operacji MINUS powoduje zastąpienie — na stosie parametrów — danej słowowej a taką daną słowową b, że $b = -a$.

ABS (a -- b)
Wykonanie operacji ABS powoduje zastąpienie — na stosie parametrów — danej słowowej a taką daną słowową b, że $b = |a|$.

MOD (a b -- c)
Wykonanie operacji MOD powoduje zastąpienie — na stosie parametrów — danych słowowych a i b taką daną słowową c, że $c = a \bmod b$.

Przykłady

a. Wyznaczenie wartości wyrażenia $w = \frac{n(n+1)}{2}$ zgodnie ze schematem (n -- w)

DUP 1+ * 2 /

b. Wyznaczenie wartości wyrażenia $w = x^2 + px + q$ zgodnie ze schematem (p q x -- w)

ROT OVER + * +

c. Wyznaczenie reszty z dzielenia liczby 1024 przez 7

1024 7 MOD □

Operacje arytmetyczne dwusłowowe

Operacje arytmetyczne dwusłowowe służą do wykonywania działań arytmetycznych na danych dwusłowowych znajdujących się na stosie parametrów. Podstawowymi operacjami tej grupy są: D+ (dodaj-dane-dwusłowowe), DMINUS (zmień-znak-danej-dwusłowowej), DABS (wyznacz-wartość-bezwzględną-danej-dwusłowowej).

D+ (a b -- c)
Wykonanie operacji D+ powoduje zastąpienie — na stosie parametrów — danych dwusłowowych a i b ich dwusłowową sumą $c = a + b$.

DMINUS (a -- b)
Wykonanie operacji DMINUS powoduje zastąpienie — na stosie parametrów — danej dwusłowowej a taką daną dwusłowową b, że $b = -a$.

DABS (a -- b)
Wykonanie operacji DABS powoduje zastąpienie — na stosie parametrów — danej dwusłowowej a taką daną dwusłowową b, że $b = |a|$.

Przykłady

a. Zwiększenie o 2 danej dwusłowowej znajdującej się na szczycie stosu parametrów

2 0 D+

b. Zwiększenie o 2 danej dwusłowowej znajdującej się na stosie parametrów poniżej danej słowowej

ROT ROT 2 0 D+ ROT 

Operacje pamięciowe

Operacje pamięciowe służą do wykonywania działań dotyczących dowolnych obszarów pamięci operacyjnej. Podstawowymi operacjami tej grupy są: @ (pobierz-słowo), ! (zapamiętaj-słowo), C@ (pobierz-znak), C! (zapamiętaj-znak), CMOVE (przepisz-ciąg-znaków), FILL (wypełnij-ciągiem-znaków).

@ (*a* -- *b*)

Wykonanie operacji @ powoduje zastąpienie — na stosie parametrów — adresu *a* danej słowowej tą daną słowową *b*.

! (*b a* --)

Wykonanie operacji ! powoduje umieszczenie pod adresem *a* danej słowowej *b*.

C@ (*a* -- *b*)

Wykonanie operacji C@ powoduje zastąpienie — na stosie parametrów — adresu *a* danej bajtowej daną słowową *b*, której górny bajt ma wartość 0, a dolny jest identyczny z bajtem spod podanego adresu *a*.

C! (*b a* --)

Wykonanie operacji C! powoduje umieszczenie pod adresem *a* danej bajtowej reprezentowanej w dolnym bajcie danej słowowej *b*.

CMOVE (*a b n* --)

Wykonanie operacji CMOVE powoduje przepisanie bajt po bajcie *n* bajtów z pola pamięci operacyjnej rozpoczynającego się od adresu *a* do pola zaczynającego się od adresu *b*.

FILL (*a n b* --)

Wykonanie operacji FILL powoduje umieszczenie w polu pamięci operacyjnej, rozpoczynającym się od adresu *a*, *n* znaków identycznych ze znakiem reprezentowanym w dolnym bajcie danej słowowej *b*.

Przykłady

a. Zastąpienie litery, występującej w pamięci operacyjnej pod adresem 2000, literą następną

```
2000 DUP C@ 1+ SWAP C!
```

b. Umieszczenie w polu pamięci operacyjnej, zaczynającym się od adresu 300, 19 znaków identycznych ze znakiem znajdującym się pod adresem 300.

```
300 DUP 1+ 19 CMOVE
```

□

Relacje

Relacje są operacjami wykorzystywanymi do porównywania danych. Podstawowymi relacjami są: < (mniejsze-niż), = (równe), > (większe-niż), 0= (równe-zero), 0< (mniejsze-niż-zero).

< (*a b* -- *t*)
(*a b* -- *f*)
Wykonanie operacji < powoduje wyznaczenie relacji $a < b$ i zastąpienie — na stosie parametrów — danych *a* i *b* daną *t* o wartości 1 — jeśli relacja jest prawdziwa, albo daną *f* o wartości 0 — w przeciwnym razie.

= (*a b* -- *t*)
(*a b* -- *f*)
Wykonanie operacji = powoduje wyznaczenie relacji $a = b$ i zastąpienie — na stosie parametrów — danych *a* i *b* daną *t* o wartości 1 — jeśli relacja jest prawdziwa, albo daną *f* o wartości 0 — w przeciwnym razie.

> (*a b* -- *t*)
(*a b* -- *f*)
Wykonanie operacji > powoduje wyznaczenie ralacji $a > b$ i zastąpienie — na stosie parametrów — danych *a* i *b* daną *t* o wartości 1 — jeśli relacja jest prawdziwa, albo daną *f* o wartości 0 — w przeciwnym razie.

0= (*a* -- *t*)
(*a* -- *f*)
Wykonanie operacji 0= powoduje wyznaczenie relacji $a = 0$ i zastąpienie — na stosie parametrów — danej *a* daną *t* o wartości 1 — jeśli relacja jest prawdziwa, albo daną *f* o wartości 0 — w przeciwnym razie.

0< (*a* -- *t*)
(*a* -- *f*)
Wykonanie operacji 0< powoduje wyznaczenie ralacji $a<0$ i zastą-

pienie — na stosie parametrów — danej *a* daną *t* o wartości 1 — jeśli relacja jest prawdziwa, albo daną *f* o wartości 0 — w przeciwnym razie.

Przykłady

a. Określenie, czy dana słowowa znajdująca się pod adresem 200 ma wartość ujemną

200 @ 0<

b. Określenie identyczności danych słowowych znajdujących się na szczycie stosu parametrów i stosu powrotów

DUP R = □

Operacje logiczne

Operacje logiczne służą do wykonywania działań logicznych. Podstawowymi operacjami tej grupy są: AND (iloczyn-logiczny), OR (suma-logiczna) i XOR (różnica-symetryczna).

AND (*a b* — — *c*)

Wykonanie operacji AND powoduje zastąpienie — na stosie parametrów — danych słowowych *a* i *b* ich iloczynem logicznym *c*, wyznaczonym równolegle na wszystkich parach odpowiadających sobie bitów danych *a* i *b*.

Zasady wyznaczania iloczynu:

bit a	bit b	bit c
0	0	0
0	1	0
1	0	0
1	1	1

OR (*a b* — — *c*)

Wykonanie operacji OR powoduje zastąpienie — na stosie parametrów — danych słowowych *a* i *b* ich sumą logiczną *c*, wyznaczoną równolegle na wszystkich parach odpowiadających sobie bitów danych *a* i *b*.

Zasady wyznaczania sumy:

bit a	bit b	bit c
0	0	0
0	1	1
1	0	1
1	1	1

XOR (*a b* — — *c*)

Wykonanie operacji XOR powoduje zastąpienie — na stosie para-

metrów — danych słowowych *a* i *b* ich różnicą symetryczną *c*, wyznaczoną równolegle na wszystkich parach odpowiadających sobie bitów danych *a* i *b*.
Zasady wyznaczania różnicy symetrycznej (sumy modulo 2):

bit a	bit b	bit c
0	0	0
0	1	1
1	0	1
1	1	0

Przykłady

a. Wyzerowanie najmniej znaczącego bitu danej bajtowej znajdującej się pod adresem 200

200 DUP C@ 254 AND SWAP C!

b. Zanegowanie bitów danej słowowej znajdującej się na szczycie stosu parametrów i wyprowadzenie jej jako liczby dziesiętnej (wykonanie operacji biernej HEX powoduje interpretowanie liczb w zapisie szesnastkowym, a wykonanie operacji biernej DECIMAL powoduje przywrócenie interpretowania liczb w zapisie dziesiętnym)

HEX FFFF XOR DECIMAL . □

Operacje słownikowe

Operacje słownikowe służą do wykonywania działań związanych z lokalizacją i umieszczaniem danych w słowniku operatorów. Do grupy tej należą także operacje dokonujące modyfikacji definicji operatorów. Podstawowymi operacjami słownikowymi są: HERE (określ-adres-szczytu-słownika), ALLOT (zmień-adres-szczytu-słownika), , (przenieś-daną-słowową-ze-stosu-parametrów-do-słownika), C, (przenieś-daną-bajtową-ze-stosu-parametrów-do-słownika), LITERAL (umieść-w-słowniku-wskazanie-operatora-i-umieść-za-nim-daną-słowową-ze-stosu-parametrów).

HERE (— — *a*)

Wykonanie operacji HERE powoduje umieszczenie — na stosie parametrów — adresu *a* pierwszego wolnego bajtu słownika.

ALLOT (*n* — —)

Wykonanie operacji ALLOT powoduje zmianę położenia pierwszego wolnego bajtu słownika o *n* bajtów (*n* może być ujemne).

, (*a* — —)

Wykonanie operacji , powoduje przeniesienie danej słowowej ze stosu parametrów do słownika.

C, (*a* --)

Wykonanie operacji C, powoduje przeniesienie do słownika danej bajtowej stanowiącej dolny bajt danej słowowej *a*.

LITERAL (*a* --)

Wykonanie operacji LITERAL powoduje umieszczenie w słowniku wskazania operatora LIT, a następnie zdjęcie – ze stosu parametrów – danej słowowej *a* i umieszczenie jej w słowniku bezpośrednio za tym wskazaniem.

Przykłady

a. Zarezerwowanie w słowniku 40 bajtów

40 ALLOT

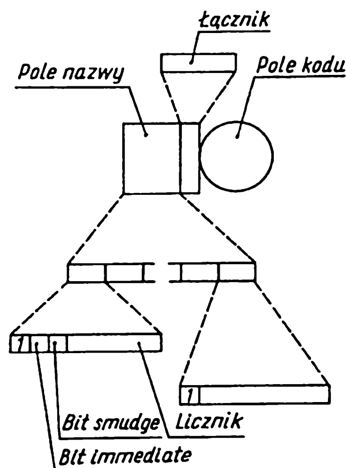
b. Umieszczenie w słowniku trzech danych bajtowych o wartościach 20, 30 i 40 (w tej kolejności)

40 30 20 C, C, C, □

4. Definiowanie operatorów

Do definiowania nowych operatorów języka służą *kompilatory*. Są to operatory, których wykonanie powoduje utworzenie w słowniku definicji nowego operatora. *Definicja operatora* składa się z *nagłówka* oraz z *pola parametrów*.

Zgodnie z rys. 4.1 nagłówek składa się z *pola nazwy, pola łącznika* oraz *pola kodu*. Pole nazwy składa się z bajtu sterującego oraz z pewnej liczby bajtów zawierających nazwę operatora przedstawioną w kodzie ASCII. Najbardziej znaczący bit pierwszego i ostatniego bajtu pola nazwy ma wartość 1. Bajt sterujący składa się zatem z bitu o wartości 1, bitu *immediate*, bitu *smudge* i 5 bitów licznika. Bit *immediate* określa, czy operator jest czynny (1) czy bierny



4.1. Nagłówek definicji operatora

(0), bit *smudge* określa, czy jest w pełni (0) czy tylko częściowo (1) zdefiniowany, a bity licznika określają liczbę znaków słowa, które określiło nazwę operatora (liczba ta może być większa od liczby znaków nazwy operatora umieszczonych w polu nazwy).

Bezpośrednio za polem nazwy występuje pole łącznika. Zawiera ono adres pola nazwy poprzedniego operatora zdefiniowanego w tym samym podsłowniku. Tuż po zainicjowaniu procesora słownik składa się zazwyczaj tylko z podsłownika Forth i w nim właśnie są umieszczane nowe definicje operatorów. Język zapewnia jednak środki do definiowania nowych podsłowników, a pola łącznika związują definicje należące do tego samego podsłownika w jednokierunkową listę. Listy dla poszczególnych podsłowników są przeglądane wstecz podczas wyszukiwania operatorów w słowniku.

Ważnym elementem nagłówka definicji jest pole kodu. Zawiera ono adres podprogramu w kodzie maszynowym. Jeśli ten adres pokrywa się z adresem pola parametrów, to wykonanie operacji związanej z tym operatorem ogranicza się do wykonania tego podprogramu. O takim operatorze mówi się, że jest *zdefiniowany w kodzie maszynowym.*

Do utworzenia nagłówka nowego operatora służy operator bierny CREATE, a do zmiany wartości bitów *immediate* i *smudge* odpowiednio operatory IMMEDIATE i SMUDGE. Wymienione uprzednio określenie „wskazanie operatora" jest w istocie adresem pola kodu definicji operatora. Skutki wykonania wymienionych tu operacji są następujące

CREATE (--)

Wykonanie operacji CREATE powoduje wprowadzenie ze strumienia wejściowego jednego słowa, a następnie utworzenie nagłówka definicji operatora o nazwie identycznej z nazwą tego słowa. Bity *immediate* i *smudge* właśnie utworzonego nagłówka otrzymają wartość 0. W polu kodu zostanie umieszczony adres bajtu następującego bezpośrednio po tym polu.

SMUDGE (--)

Wykonanie operacji SMUDGE powoduje zanegowanie bitu *smudge* ostatniej definicji słownika.

IMMEDIATE (--)

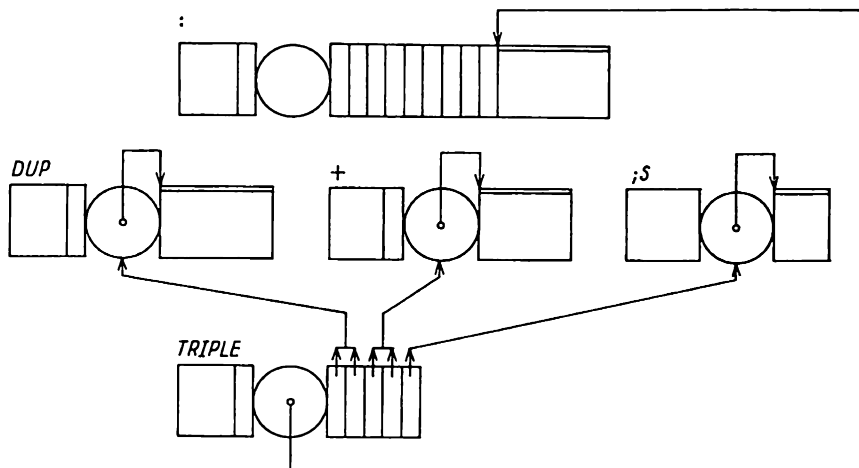
Wykonanie operacji IMMEDIATE powoduje zanegowanie bitu *immediate* ostatniej definicji słownika.

Znacznie częściej od operatora CREATE, do definiowania nowych operatorów jest stosowany operator : (dwukropek) i związany z nim operator ; (średnik). Słowo występujące bezpośrednio za operatorem : określa nazwę nowego, właśnie definiowanego operatora, a dalsze słowa, aż do ; wyłącznie

stanowią treść definicji. W tym sensie zinterpretowanie sekwencji słów

: TRIPLE
DUP DUP + + ;

powoduje utworzenie w słowniku definicji operatora TRIPLE. Przyszłe wykonanie operacji TRIPLE spowoduje dwukrotne wykonanie operacji DUP i dwukrotne wykonanie operacji +, tj. potrojenie wartości danej znajdującej się na szczycie stosu parametrów. Na rysunku 4.2 przedstawiono definicję



4.2. Struktura definicji operatora TRIPLE

słownikową operatora TRIPLE i jego powiązania z operatorami :, DUP, + i ;S. Na rysunku tym pole nazwy wraz z polem łącznika przedstawiono w postaci kwadratu, pole kodu w postaci okręgu, wskazania jako pionowe prostokąty, a podprogramy zapisane w kodzie maszynowym – jako prostokąty wyróżnione poziomą kreską. Zgodnie z tą definicją, pole kodu operatora TRIPLE zawiera adres podprogramu umieszczonego w polu parametrów definicji operatora :, a pole parametrów definicji operatora TRIPLE składa się ze wskazań operatorów DUP, + i ;S. Pola kodu tych operatorów zawierają adresy ich pól parametrów. Oznacza to, że wymienione operatory są zdefiniowane w kodzie maszynowym.

W ogólnym przypadku definicja nowego operatora ma postać

: *nazwa*
słowa ;

gdzie *nazwa* jest argumentem operatora : (dwukropek), słowa zaś określają treść definicji operatora *nazwa*. Ponieważ wykonanie operacji : powoduje ustawienie interpretera w stan definiowania, poszczególne operatory sekwencji

słowa nie są interpretowane, a jedynie — bezpośrednio za nagłówkiem definicji operatora *nazwa* — są umieszczane w słowniku wskazania tych operatorów. Wyjątek w tym względzie stanowią jedynie operatory czynne, które są interpretowane również w stanie definiowania. Takim operatorem jest np. ; (średnik), którego zinterpretowanie powoduje umieszczenie w słowniku wskazania operatora ;S i ustawienie interpretera w stan wykonywania. Tym samym wykonanie operacji : (dwukropek) powoduje przełączenie interpretera w stan kompilacji jedynie na czas definiowania operatora *nazwa*. Operator *nazwa* zostaje włączony do słownika, a jego definicja — połączona z definicjami innych operatorów.

Ponieważ wykonanie operacji : (dwukropek) powoduje nadanie bitowi *smudge* operatora *nazwa* wartości 1, bezpośrednio po wykonaniu tej operacji operator *nazwa* jest traktowany tak, jakby nie był w pełni zdefiniowany. Umożliwia to odwoływanie się w sekwencji *słowa* do operatora *nazwa* w pełni zdefiniowanego już uprzednio. Dopiero po wykonaniu operacji ; (średnik), kiedy to nastąpi zanegowanie bitu *smudge*, nowa definicja przesłoni następną. Jak z tego wynika, mechanizm związany z bitem *smudge* umożliwia m.in. zdefiniowanie operatora odwołującego się do wcześniej zdefiniowanego operatora o tej samej nazwie.

Jeśli definiowany operator ma należeć do kategorii operatorów czynnych, to po zdefiniowaniu jego nagłówka należy posłużyć się operatorem IMMEDIATE. Następuje to najczęściej po wykonaniu operacji ; (średnik). Jak już wyjaśniano, operacja czynna jest wykonywana nawet wtedy, gdy interpreter znajduje się w stanie definiowania.

W tych przypadkach, gdy jest wymagane umieszczenie w słowniku wskazania operatora czynnego, a nie wykonanie operacji związanej z tym operatorem, należy poprzedzić go specjalnym operatorem czynnym [COMPILE]. Natomiast wtedy, gdy podczas definiowania wymagane jest wykonanie pewnej sekwencji operacji biernych należy poprzedzić ją operatorem czynnym [i zakończyć operatorem biernym]. W istocie, wykonanie operacji [powoduje po prostu ustawienie interpretera w stan wykonywania, a wykonanie operacji] powoduje ustawienie go w stan definiowania.

Niekiedy zachodzi potrzeba zdefiniowania takiej operacji, której wykonanie spowoduje umieszczenie w słowniku wskazania określonego operatora. Do tego celu służy operacja bierna COMPILE. Jej wykonanie powoduje umieszczenie w słowniku takiego samego wskazania, jakie występuje po wskazaniu operatora COMPILE.

Przykłady

a. Definiowanie operatora biernego

```
: ZERO DROP 0 ;
```

• Wykonanie operacji ZERO powoduje zastąpienie danej znajdującej się na szczycie stosu parametrów daną o wartości 0.

• W polu parametrów operatora ZERO występują: wskazanie operatora DROP, wskazanie operatora LIT, dana o wartości 0, wskazanie operatora ;S.

• Wykonanie sekwencji operacji

30 : ADD 20 ZERO + ; ADD

spowoduje zdefiniowanie operatora ADD oraz umieszczenie — na stosie parametrów — danej o wartości 30. W polu parametrów operatora ADD znajdą się: wskazanie operatora LIT, dana słowowa o wartości 20, wskazanie operatora ZERO, wskazanie operatora + (plus) i wskazanie operatora ;S.

b. Definiowanie operatora czynnego

: ZERO DROP 0 ; IMMEDIATE

• Wykonanie operacji ZERO powoduje zastąpienie danej znajdującej się na szczycie stosu parametrów daną o wartości 0. Operacja ZERO jest wykonywana również w stanie definiowania.

• W polu parametrów operatora ZERO występują: wskazanie operatora DROP, wskazanie operatora LIT, dana słowowa o wartości 0 i wskazanie operatora ;S.

o Wykonanie sekwencji operacji

30 : ADD 20 ZERO + ; ADD

spowoduje zdefiniowanie operatora oraz umieszczenie — na stosie parametrów — danej o wartości 20. W polu parametrów operatora ADD znajdą się: wskazanie operatora LIT, dana słowowa o wartości 20 i wskazanie operatora +. Nie wystąpi tam wskazanie operatora ZERO, ponieważ zinterpretowanie słowa ZERO spowodowało natychmiastowe wykonanie operacji ZERO, a nie umieszczenie w słowniku wskazania operatora ZERO. Wynika to stąd, że ZERO jest operatorem czynnym.

c. Użycie operatora [COMPILE]

: GET DUP ; IMMEDIATE

: NEG [COMPILE] GET MINUS ;

• Wykonanie operacji GET powoduje powielenie — na stosie parametrów — danej słowowej znajdującej się na jego szczycie. Operacja GET jest wykonywana także wtedy, gdy interpreter znajduje się w stanie definiowania.

• Wykonanie operacji NEG powoduje umieszczenie — na stosie parametrów — dwójkowego uzupełnienia danej słowowej znajdującej się na jego szczycie.

• W polu parametrów operatora NEG występują: wskazanie operatora GET, wskazanie operatora MINUS i wskazanie operatora ;S.

• Gdyby w definicji operatora NEG nie wystąpiło słowo [COMPILE], to pole parametrów tego operatora nie zawierałoby wskazania operatora GET.

d. Użycie operatorów [i]

: VALUE [2 5 +] LITERAL ;

• Operatory występujące między [i] są traktowane tak, jakby były czynne.

• Ponieważ LITERAL jest operatorem czynnym, wykonanie operacji LITERAL powoduje umieszczenie w słowniku wskazania operatora LIT oraz przeniesienie do słownika danej słowowej znajdującej się na szczycie stosu parametrów. W polu parametrów operatora VALUE wystąpią: wskazanie operatora LIT, dana słowowa o wartości 7 i wskazanie operatora ;S.

• Gdyby w definicji operatora VALUE opuszczono operator], to stałaby się ona błędna, ponieważ podczas interpretowania operatora ; (średnik) stwierdzono by, że zinterpretowanie sekwencji operacji zawartych między operatorem : (dwukropek) a operatorem ; (średnik) spowodowało zmianę rozmiaru stosu parametrów.

e. Użycie operatora COMPILE

```
: OVER2 COMPILE OVER COMPILE OVER ;
  IMMEDIATE
: ADD OVER2 + ;
```

• Wykonanie operacji OVER2 powoduje umieszczenie w słowniku dwóch następujących po sobie wskazań operatora OVER.

• Wykonanie operacji ADD powoduje umieszczenie — na stosie parametrów — słowowej sumy danych słowowych znajdujących się u szczytu stosu parametrów.

• W polu parametrów operatora ADD występują: dwa następujące po sobie wskazania operatora OVER, wskazanie operatora + (plus) i wskazanie operatora ;S.

• Gdyby operator OVER2 zdefiniowano jako bierny, to w polu parametrów operatora ADD wystąpiłyby: wskazanie operatora OVER2, wskazanie operatora + i wskazanie operatora ;S.

f. Złożone użycie operatorów [COMPILE] i COMPILE

```
: GET R ; IMMEDIATE
: SET COMPILE [COMPILE] GET ; IMMEDIATE
: NEG SET MINUS ;
```

• W polu parametrów operatora SET występują: wskazanie operatora COMPILE, wskazanie operatora GET i wskazanie operatora ;S.

• W polu parametrów operatora NEG występują: wskazanie operatora GET, wskazanie operatora MINUS i wskazanie operatora ;S.

g. Zdefiniowanie czynnego operatora-liczby

```
: -5 -5 ; IMMEDIATE
```

• Po zinterpretowaniu przytoczonej sekwencji −5 jest operatorem czynnym.

• Wykonanie operacji −5 powoduje umieszczenie — na stosie para-

metrów — danej słowowej o wartości −5. Dzieje się tak nawet wtedy, gdy interpreter znajduje się w stanie definiowania.

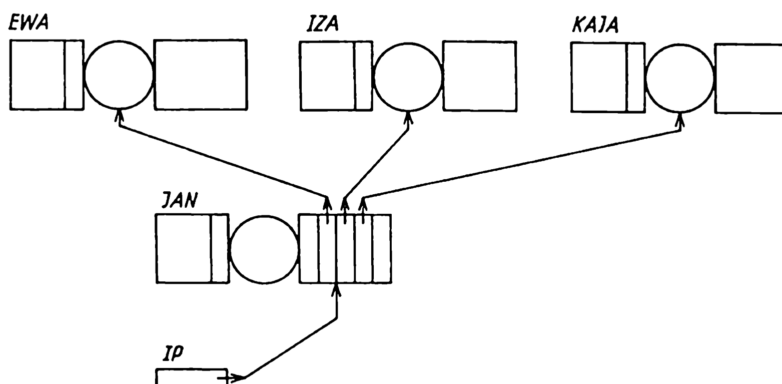
• Gdyby −5 zdefiniowano jako operator bierny, to skutek zinterpretowania go byłby taki sam, jak skutek zinterpretowania liczby −5. Wykonanie operacji −5 byłoby jednak szybsze. □

5. Stos powrotów

Podobnie jak stos operatorów, *stos powrotów* jest strukturą typu LIFO, której elementami są dane słowowe. Stos powrotów może być wykorzystywany jako pomocniczy stos roboczy, na którym są przechowywane dane słowowe oraz parametry niektórych instrukcji strukturalnych. Jego głównym zadaniem jest jednak przechowywanie śladu wzajemnych wywołań operatorów. Zbiór operacji dotyczących stosu powrotów jest nader skromny. Ogranicza się on do omówionych już operacji R, >R, R>, RP!, RP@ i R0.

W celu bliższego wyjaśnienia zasad posługiwania się stosem powrotów konieczne jest odwołanie się do pojęcia *licznik interpretera*. Licznikiem tym, oznaczanym IP, jest pewien wewnętrzny rejestr procesora, który w chwili rozpoczęcia interpretowania definicji operatora zawiera adres wskazania pola kodu tego operatora, który w zwykłych warunkach będzie interpretowany jako następny. Pokazano to na rys. 5.1, z którego wynika, że jeśli podczas interpretowania operatora JAN zostanie napotkane w słowniku wskazanie operatora EWA, to w chwili podjęcia czynności związanych z interpretacją operatora EWA, licznik interpretera będzie zawierał adres wskazania pola kodu operatora IZA. Spowoduje to, że w zwykłych warunkach, bezpośrednio po wykonaniu operacji EWA, będzie kontynuowane wykonywanie operacji JAN, a w chwili podjęcia interpretowania operacji IZA licznik interpretera będzie zawierał adres wskazania operatora KAJA.

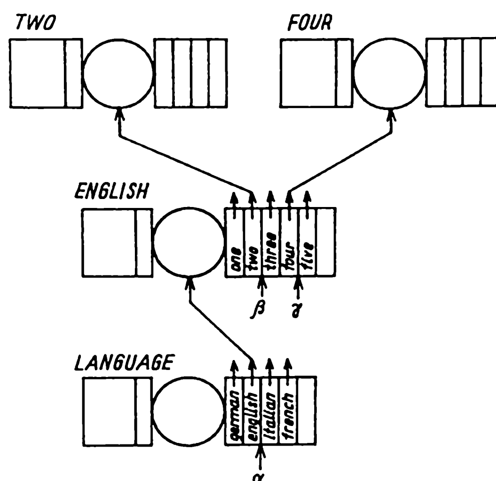
Jeśli operatory takie jak EWA, IZA i KAJA są zdefiniowane w kodzie maszynowym, to w każdym przypadku ich pole kodu zawiera adres pierwszego bajtu pola parametrów. W polu tym znajduje się kod maszynowy, a po jego wykonaniu następuje zwiększenie licznika interpretera o 2. Spowoduje to przejście do interpretowania następnego operatora, np. operatora KAJA bezpośrednio po operatorze IZA.



5.1. Licznik interpretera

Jeśli pewien operator jest zdefiniowany za pomocą kompilatora : (dwukropek), to pole kodu tego operatora zawiera adres podprogramu w kodzie maszynowym, którego wykonanie powoduje m.in. przeniesienie zawartości licznika intepretera na stos powrotów. Po zakończeniu wykonywania operacji definiujących taki operator, dana ze szczytu stosu powrotów zostaje przeniesiona do licznika interpretera. Przebieg dalszej interpretacji zależy od tej nowej wartości licznika. Należy nadmienić, że podczas interpretowania operatora wartość ta może ulec zmianie.

W celu podsumowania omówionych tu zasad posługiwania się stosem powrotów, na rys. 5.2 przedstawiono strukturę powiązań kilku operatorów



5.2. Zasady posługiwania się stosem powrotów

zdefiniowanych za pomocą kompilatora : (dwukropek). Przyjęto dla uproszczenia, że wykonanie nie zdefiniowanych jawnie operatorów nie ma skutków ubocznych takich jak np. zmiana danych na stosie powrotów.

```
: TWO  R>  2+  >R  ;
: FOUR  R>  DROP  TWO  ;
. . .
: ENGLISH  ONE  TWO  THREE  FOUR  FIVE  ;
. . .
: LANGUAGE  GERMAN  ENGLISH  ITALIAN  FRENCH  ;
```

Zgodnie z przytoczonymi uprzednio zasadami wykorzystania stosu powrotów można podać, że

- Podczas wykonywania operacji ENGLISH, realizowanej w ramach wykonania operacji LANGUAGE, na szczycie stosu powrotów znajduje się adres α wskazania operatora ITALIAN.
- Podczas wykonywania operacji TWO, realizowanej w ramach wykonania operacji ENGLISH, na szczycie stosu powrotów znajduje się adres β wskazania operatora THREE.
- Ponieważ wykonanie operacji TWO powoduje zwiększenie śladu na stosie powrotów o 2, bezpośrednio po wykonaniu operacji TWO nastąpi wykonanie operacji FOUR, tj. ominięcie operacji THREE.
- Podczas wykonywania operacji FOUR, realizowanej w ramach wykonania operacji ENGLISH, na szczycie stosu powrotów znajduje się adres γ wskazania operatora FIVE.
- Ponieważ wykonanie operacji FOUR powoduje usunięcie ze stosu powrotów śladu γ oraz zwiększenie o 2 śladu α, bezpośrednio po wykonaniu operacji FOUR nastąpi wykonanie operacji FRENCH, tj. ominięcie zarówno operacji FIVE jak i operacji ITALIAN.

Przytoczone zasady umieszczania na stosie powrotów śladu wywołań operatorów dotyczą jedynie sytuacji, gdy są realizowane operacje zdefiniowane za pomocą kompilatora : (dwukropek). Przejście do wykonania operacji zdefiniowanych w kodzie maszynowym nie powoduje bowiem żadnych zmian na stosie powrotów.

Przykład

Wpływ sposobu zdefiniowania operacji na stan stosu powrotów

```
: IBM  PERSONAL  MAIN-FRAME  ;
: COMPUTER  DEC  IBM  CDC  ;
```

- W typowych warunkach wykonanie operacji COMPUTER spowoduje wykonanie operacji IBM, a w ramach wykonania operacji IBM spowoduje wykonanie operacji PERSONAL.

● Jeśli operacja PERSONAL jest zdefiniowana za pomocą kompilatora : (dwukropek), to podczas jej wykonywania na stosie powrotów znajduje adres wskazania operatora MAIN—FRAME.

● Jeśli operacja PERSONAL jest zdefiniowana w kodzie maszynowym, to podczas jej wykonania na stosie powrotów znajduje się adres wskazania operatora CDC. □

6. Definiowanie kompilatorów

Kompilatorem jest operator związany z taką operacją, której wykonanie powoduje utworzenie definicji nowego operatora. Poza omówionymi już kompilatorami : (dwukropek) i CREATE, predefiniowanymi kompilatorami języka Forth są m.in. CONSTANT i VARIABLE.

CONSTANT (*v* --)

Wykonanie operacji CONSTANT powoduje utworzenie definicji operatora o takiej samej nazwie, jaką ma najbliższe słowo w strumieniu wejściowym oraz umieszczenie w polu parametrów tego operatora danej słowowej o wartości *v*. Przyszłe wykonanie operacji związanej z tak utworzonym operatorem spowoduje umieszczenie — na stosie parametrów — danej o wartości *v*.

VARIABLE (*v* --)

Wykonanie operacji VARIABLE powoduje utworzenie definicji operatora o takiej samej nazwie, jaką ma najbliższe słowo w strumieniu wejściowym oraz umieszczenie w polu parametrów tego operatora danej słowowej o wartości *v*. Przyszłe wykonanie operacji związanej z tak utworzonym operatorem spowoduje umieszczenie — na stosie parametrów — adresu wspomnianej danej.

Podobnie jak do zdefiniowania operatora można było posłużyć się sekwencją

: *nazwa*
  słowa-definiujące ;

tak w szczególności do zdefiniowania kompilatora można posłużyć się sekwencją

```
: nazwa
  <BUILDS
        słowa-kompilujące
  DOES>
        słowa-wykonawcze
;
```

W sekwencji tej *nazwa* identyfikuje kompilator, *słowa-kompilujące* określają czynności wykonywane przez kompilator podczas tworzenia nowej definicji, a *słowa-wykonawcze* określają czynności realizowane podczas wykonywania operacji związanej z definicją operatora utworzoną przez dany kompilator. Dzięki odpowiedniemu dobraniu definicji operatorów <BUILDS i DOES>, wykonanie operacji *nazwa* nie powoduje wykonania czynności określonych przez *słowa-wykonawcze*. Czynności te są realizowane dopiero podczas wykonywania operacji związanej z definicją operatora utworzoną za pomocą kompilatora *nazwa*.

Zarówno <BUILDS, jak i DOES> są operatorami biernymi, zdefiniowanymi za pomocą kompilatora : (dwukropek). Ich zinterpretowanie powoduje wykonanie następujących czynności.

<BUILDS (--)

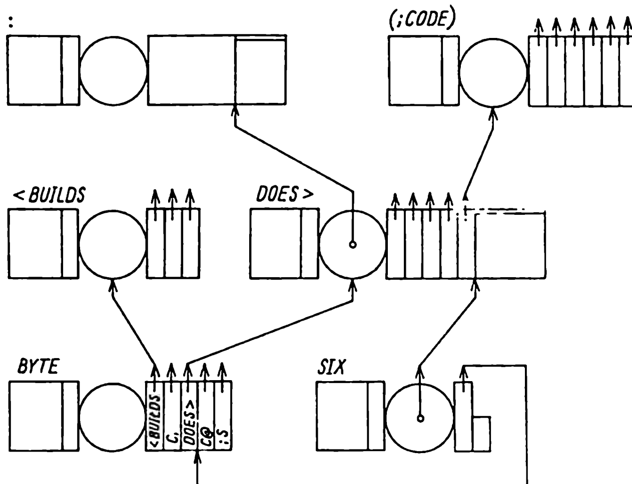
Wykonanie operacji biernej <BUILDS powoduje utworzenie nagłówka definicji operatora o takiej samej nazwie, jaką ma najbliższe słowo w strumieniu wejściowym oraz umieszczenie w polu parametrów tego operatora danej słowowej o wartości 0.

DOES> (--)

Wykonanie operacji biernej DOES> powoduje zakończenie interpretowania definicji tego operatora, w której wystąpiło odwołanie do DOES>. Zanim to nastąpi, w polu parametrów ostatnio zdefiniowanego operatora zostanie umieszczony ślad zdjęty ze stosu powrotów, tj. adres *słów-wykonawczych* następujących po wskazaniu operatora DOES>, a w polu kodu wspomnianego operatora zostanie umieszczony adres podprogramu w kodzie maszynowym. Podprogram ten jest wspólny dla wszystkich kompilatorów zdefiniowanych za pomocą operatora DOES> i jest tak dobrany, że jego wykonanie powoduje umieszczenie — na stosie parametrów — adresu trzeciego bajtu pola parametrów operatora i przygotowanie interpretera do wykonania *słów-wykonawczych*.

Na rysunku 6.1 przedstawiono strukturę definicji przykładowego kompilatora BYTE zdefiniowanego za pomocą sekwencji

```
: BYTE
  <BUILDS  C,  DOES>  C@  ;
```



6.1. Struktura definicji kompilatora BYTE i operatora SIX

oraz operatora SIX skompilowanego przez BYTE w następstwie zinterpretowania sekwencji

6 BYTE SIX

Po wykonaniu tej ostatniej czynności w słowniku zostaje umieszczona definicja operacji SIX. Przyszłe wykonanie tej operacji spowoduje umieszczenie na stosie parametrów danej słowowej, której górny bajt ma wartość 0, a dolny pochodzi z trzeciego bajtu pola parametrów operatora SIX. Łatwo zauważyć, że w ogólnym przypadku użycie kompilatora BYTE powoduje definiowanie operatorów identyfikujących stałe bajtowe. Jak wynika z przytoczonego opisu, wykonanie kompilacji

6 BYTE SIX

składa się z wykonania operacji <BUILDS (utworzenie nagłówka definicji operatora SIX) oraz z wykonania operacji C, (przeniesienie danej bajtowej z pola parametrów do słownika). Wykonanie operacji DOES> powoduje zakończenie czynności związanych z definiowaniem operatora SIX i zaniechanie interpretowania *słów-wykonawczych*, tu C@. Słowa te są interpretowane podczas wykonywania operacji SIX. Zanim to nastąpi, na stosie parametrów zostaje umieszczony adres trzeciego bajtu pola parametrów, tj. tego bajtu, w którym podczas kompilacji umieszczono daną bajtową o wartości 6.

Przykłady

a. Użycie kompilatora CONSTANT

400 CONSTANT ALPHA

• Wykonanie operacji ALPHA powoduje umieszczenie — na stosie parametrów — danej słowowej o wartości 400.

b. Użycie kompilatora VARIABLE

```
400 VARIABLE BETA
```

• Wykonanie operacji BETA powoduje umieszczenie — na stosie parametrów — adresu danej o wartości 400.

c. Zdefiniowanie kompilatora tablic jednowymiarowych

```
: VECTOR
  <BUILDS ALLOT
   DOES> + ;
```

• Wykonanie operacji VECTOR (*n* --) powoduje utworzenie definicji operatora, w którego polu parametrów zostanie zarezerwowany obszar o rozmiarze *n* bajtów.

• Wykonanie sekwencji

```
5 VECTOR BUF
```

powoduje zdefiniowanie operatora BUF, w którego polu parametrów występuje obszar roboczy o rozmiarze 5 bajtów.

• Wykonanie sekwencji

```
13 2 BUF C!
```

powoduje umieszczenie w środkowym bajcie obszaru roboczego danej bajtowej o wartości 13.

d. Zdefiniowanie kompilatora tablic dwuwymiarowych

```
: ARRAY
  <BUILDS OVER C, DUP C, * ALLOT
   DOES> 1+ ROT OVER C@ * + + 1+ ;
```

• Wykonanie operacji ARRAY (*w k* --) powoduje utworzenie definicji operatora, w którego polu parametrów zostaną umieszczone dane bajtowe *w* i *k*, określające rozmiary tablicy. Ponadto zostanie zarezerwowany obszar roboczy o rozmiarze $w * k$ bajtów.

• Wykonanie sekwencji

```
2 3 ARRAY BUF
```

powoduje zdefiniowanie bufora BUF, w polu parametrów którego występują dane bajtowe o wartościach 2 i 3 oraz obszar o rozmiarze 6 bajtów. W obszarze tym znajduje się miejsce dla elementów tablicy. Przyjmując, że indeksy wierszy i kolumn są liczone od zera, a elementy tablic są uporządkowane wierszami, można podać, że element o indeksie (i, j) znajduje się w bajcie o numerze $i*k+j$.

• Wykonanie sekwencji

```
13 2 1 BUF C!
```

powoduje przypisanie elementowi tablicy o indeksie (2,1) danej bajtowej o wartości 13.

e. Zdefiniowanie kompilatora operatorów wyboru

```
: SELECT
  <BUILDS SMUDGE ]
   DOES> SWAP 2 * + @ EXECUTE ;
```

• Wykonanie użytej tu operacji EXECUTE (*a* — —) powoduje wykonanie operacji, której pole kodu znajduje się pod adresem *a*.

• Jeśli PRINT, ZERO i ADD są przykładowymi operatorami o definicjach

```
: PRINT ". JANEK" ;
: ZERO DROP 0 ;
ADD + + ;
```

to zinterpretowanie sekwencji

```
SELECT OBEY PRINT ZERO ADD ;
```

spowoduje utworzenie takiej definicji operatora OBEY, że sekwencja 0 OBEY jest wykonywana tak jak operator PRINT, sekwencja 1 OBEY jest wykonywana tak jak operator ZERO, a sekwencja 2 OBEY jest wykonywana tak jak operator ADD.

• Użycie w definicji kompilatora SELECT operatora] powoduje przejście interpretera w stan definiowania. Powoduje to, że aż do napotkania operatora czynnego ; (przywracającego stan wykonywania), w definicji operatora OBEY są umieszczane wskazania operatorów PRINT, ZERO i ADD.

• Ponieważ wykonanie operacji ; kończącej sekwencję operatorów PRINT ZERO ADD powoduje zanegowanie bitu *smudge* operatora OBEY, w definicji kompilatora SELECT posłużono się operacją SMUDGE kompensująca to zanegowanie.

7. Instrukcje strukturalne

W ślad za wieloma innymi językami programowania, także w języku Forth zdefiniowano *instrukcje strukturalne*. Instrukcje te mają postać sekwencji operatorów, wśród których występują *operatory kluczowe*, jak IF, THEN, WHILE itp., wytyczające zakresy poszczególnych instrukcji strukturalnych.

Instrukcje strukturalne mogą być używane jedynie w tych sekwencjach operatorów, podczas interpretowania których interpreter znajduje się w stanie definiowania. Z tego względu wszystkie operatory kluczowe należą do kategorii operatorów czynnych.

Instrukcja warunkowa

Instrukcja ta ma postać

```
przed-if IF
  po-if-przed-else
ELSE
  po-else-przed-then
THEN
```

gdzie *przed-if*, *po-if-przed-else* i *po-else-przed-then* są dowolnymi sekwencjami operatorów.

Po wykonaniu sekwencji *przed-if* następuje zdjęcie — ze stosu parametrów — danej słowowej i rozpatrzenie jej wartości. Jeśli wartość ta jest różna od 0, to jest wykonywana sekwencja *po-if-przed-else*. W przeciwnym razie jest wykonywana sekwencja *po-else-przed-then*.

W każdym kontekście operator kluczowy THEN może być zastąpiony operatorem kluczowym ENDIF, a w przypadku gdy sekwencja *po-else-przed-then* jest pusta, może być opuszczony operator kluczowy ELSE.

Przykłady

a. Wykorzystanie instrukcji warunkowej

```
: INP
  KEY KEY OVER OVER =
  IF
    ." BOTH " EMIT DROP
  ELSE
    SWAP EMIT EMIT
  THEN ;
```

• Wykonanie operacji INP powoduje wprowadzenie z klawiatury terminala dwóch znaków i porównanie ich.

• Jeśli znaki są identyczne, to jest wyprowadzany napis BOTH oraz jeden ze znaków.

• Jeśli znaki są różne, to są one wyprowadzane w kolejności ich wprowadzenia.

b. Wykorzystanie instrukcji warunkowej bez ELSE

```
: CHECK ( -- )
  DUP 0<
  IF
    ." TOP OF STACK NEGATIVE"
  THEN ;
```

• Wykonanie operacji CHECK powoduje rozpatrzenie wartości danej słowowej znajdującej się na szczycie stosu parametrów.

• Jeśli dana ta ma wartość ujemną, to jest wyprowadzany napis TOP OF STACK NEGATIVE.

• W przeciwnym razie nie dzieje się nic.

c. Wykorzystanie instrukcji warunkowej do zdefiniowania operacji rekurencyjnej

```
: FACTORIAL
  DUP
  IF
    DUP 1 -
    [ SMUDGE ] FACTORIAL [ SMUDGE ]
    *
  ELSE
    1+
  THEN ;
```

• Wykonanie operacji FACTORIAL powoduje zastąpienie — na stosie parametrów — nieujemnej danej słowowej *n* jej słowową silnią *n*!.

• Wykonanie operacji SMUDGE przed i po odwołaniu do operacji FACTORIAL powoduje, że odwołanie to dotyczy właśnie definiowanego operatora. □

Instrukcja iteracyjna DO ... LOOP

Instrukcja ta występuje w kontekście

przed-do
DO
    po-do-przed-loop
LOOP
po-loop

gdzie *przed-do, po-do-przed-loop* i *po-loop* są dowolnymi sekwencjami operatorów i instrukcji.

Po wykonaniu sekwencji *przed-do* następuje ustalenie dwóch parametrów: ograniczenia *l* oraz wartości początkowej *i*, zmiennej sterującej cyklu. Po wykonaniu tych czynności następuje cykliczne wykonywanie sekwencji *po-do-przed-loop* dla zmiennej sterującej cyklu przybierającej wartości od *i* do *l* wyłącznie, z krokiem +1. Jeśli podczas wykonywania cyklu zostanie wykonana operacja I, to na szczycie stosu parametrów zostanie umieszczona dana słowowa określająca bieżącą wartość zmiennej sterującej cyklu. Po zakończeniu cyklu następuje przejście do wykonywania sekwencji *po-loop*. Ponieważ rozstrzyganie o kontynuowaniu cyklu odbywa się na końcu jego zakresu, tj. po wykonaniu sekwencji *po-do-przed-loop*, cykl jest wykonywany co najmniej jednokrotnie, a więc nawet wtedy gdy wartość początkowa zmiennej sterującej cyklu przekracza wartość ograniczenia.

Przykłady

a. Wykonanie cyklu iteracyjnego

```
: OUT
  7 3 DO I . LOOP ;
```

- Wykonanie operacji OUT spowoduje wyprowadzenie liczb 3, 4, 5 i 6.
- Zakres cyklu zawiera odwołania do operacji I i . (kropka).

b. Wykonanie cyklu zawartego w innym cyklu

```
: OUT
  4 1
  DO
     CR I 4 1
     DO
        DUP 1 .R I .
     LOOP
     DROP
  LOOP ;
```

• Wykonanie operacji OUT powoduje wyprowadzenie napisu

11 12 13
21 22 23
31 32 33

• Pierwsze odwołanie do I dotyczy cyklu zewnętrznego, a drugie dotyczy cyklu wewnętrznego. □

Instrukcja iteracyjna DO ... +LOOP

Instrukcja ta występuje w kontekście

przed-do
DO
po-do-przed-loop
+LOOP
po-loop

gdzie *przed-do*, *po-do-przed-loop* i *po-loop* są dowolnymi sekwencjami operatorów i instrukcji.

Po wykonaniu sekwencji *przed-do* następuje ustalenie dwóch parametrów: ograniczenia *l* oraz wartości początkowej *i*, zmiennej sterującej cyklu. Po wykonaniu tych czynności następuje wykonanie sekwencji *po-do-przed-loop*. Następnie ze stosu parametrów jest zdejmowana dana słowowa i wartość tej danej jest dodawana do zmiennej sterującej cyklu. Jeśli dodana wartość była dodatnia, a rezultat dodawania jest mniejszy od ograniczenia, jak również wtedy gdy dodana wartość była ujemna, a rezultat jest większy od ograniczenia, wykonanie zakresu cyklu jest powtarzane. W przeciwnym razie następuje zakończenie wykonywania instrukcji strukturalnej i przejście do wykonywania sekwencji *po-loop*.

Przykłady

a. Wykonanie cyklu z krokiem dodatnim, różnym od 1

```
: OUT
  9 4 DO I . 2 +LOOP ;
```

• Wykonanie operacji OUT powoduje wyprowadzenie liczb 4, 6 i 8.

• Gdyby w przytoczonej definicji zmieniono liczbę 9 na 10, to skutek wykonania operacji OUT byłby taki sam.

b. Wykonanie cyklu z krokiem ujemnym

```
: OUT
  -3 1 DO I . -2 +LOOP ;
```

• Wykonanie operacji OUT powoduje wyprowadzenie liczb 1 i −1.

• Gdyby w przytoczonej definicji zmieniono liczbę −3 na −2, to skutek wykonania operacji OUT byłby taki sam. □

Instrukcja repetycyjna BEGIN ... WHILE ... REPEAT

Instrukcja ta występuje w kontekście

BEGIN
  po-begin-przed-while WHILE
  po-while-przed-repeat
REPEAT
po-repeat

gdzie *po-begin-przed-while, po-while-przed-repeat* i *po-repeat* są dowolnymi sekwencjami operatorów i instrukcji.

Po wykonaniu sekwencji *po-begin-przed-while* następuje zdjęcie — ze stosu parametrów — danej słowowej i rozpatrzenie jej wartości. Jeśli wartość ta jest różna od 0, to nastąpi wykonanie sekwencji *po-while-przed-repeat* i powtórzenie opisanych czynności od początku. W przeciwnym razie nastąpi zakończenie wykonywania instrukcji strukturalnej i przejście do wykonywania sekwencji *po-repeat*.

Przykład

Wykonanie cyklu repetycyjnego

```
: TALLY  ( a -- n )
  0
  BEGIN
     SWAP  DUP  1+  SWAP  C@  WHILE
     SWAP  1+
  REPEAT
  DROP  ;
```

● Przed wykonaniem operacji TALLY na szczycie stosu parametrów znajduje się adres pola pamięci operacyjnej. Pole to zawiera ciąg znaków w kodzie ASCII zakończony znakiem o kodzie 0.

● Po wykonaniu operacji TALLY na szczycie stosu parametrów znajduje się dana słowowa określająca liczbę znaków wspomnianego pola.

□

Instrukcja repetycyjna BEGIN ... UNTIL

Instrukcja ta występuje w kontekście

BEGIN
  po-begin-przed-until
UNTIL
po-until

gdzie *po-begin-przed-until* i *po-until* są dowolnymi sekwencjami operatorów i instrukcji.

Po wykonaniu sekwencji *po-begin-przed-until* następuje zdjęcie — ze stosu parametrów — danej słowowej i rozpatrzenie jej wartości. Jeśli wartość ta jest równa 0, to nastąpi powtórzenie opisanych czynności od początku. W przeciwnym razie nastąpi zakończenie wykonywania instrukcji strukturalnej i przejście do wykonywania sekwencji *po-until*.

Przykład

Wykonanie cyklu repetycyjnego

```
: TALLY  ( a -- n )
  1 - -1
  BEGIN
    1+ SWAP 1+ SWAP OVER C@ 0=
  UNTIL
  SWAP DROP ;
```

• Przed wykonaniem operacji TALLY na szczycie stosu parametrów znajduje się adres pola pamięci operacyjnej. Pole to zawiera ciąg znaków w kodzie ASCII, zakończony znakiem o kodzie 0.

• Po wykonaniu operacji TALLY na szczycie stosu parametrów znajduje się dana słowowa *n* określająca liczbę znaków wspomnianego pola. □

Instrukcja repetycyjna BEGIN ... AGAIN

Instrukcja ta występuje w kontekście

BEGIN
  po-begin-przed-again
AGAIN

gdzie *po-begin-przed-again* jest dowolną sekwencją operatorów i instrukcji.

Po wykonaniu sekwencji *po-begin-przed-again* następuje powtórzenie wykonania tych czynności od początku. Wyznaczony przez instrukcję repetycyjną cykl nieskończony może zostać przerwany, jeśli w zakresie tej instrukcji zostanie wykonana np. operacja ABORT.

Przykład

Wykonanie cyklu repetycyjnego

```
: CONVERT  ( cały-stos -- )
  -1
  BEGIN
    1+ DUP DUP CR
    6 .R HEX 6 .R DECIMAL
```

```
    ?TERMINAL IF ABORT THEN
  AGAIN ;
```

- Wykonanie operacji CONVERT powoduje wyprowadzenie bliżej nieokreślonej liczby par liczb szesnastkowych i dziesiętnych.
- Wprowadzenie z klawiatury znaku *break* powoduje wykonanie operacji ABORT, kończącej cykl repetycyjny. □

8. Definiowanie instrukcji strukturalnych

Zestaw instrukcji strukturalnych języka Forth jest na tyle obszerny, iż umożliwia efektywne formułowanie dostatecznie złożonych algorytmów. Ponieważ jednak ważną cechą tego języka jest jego rozbudowywalność, przejawiająca się w możliwości definiowania w nim nowych operatorów i instrukcji, celowe wydaje się pokazanie sposobu definiowania instrukcji strukturalnych za pomocą podstawowych operatorów języka.

Dla zilustrowania typowej metody postępowania zostanie przedstawiona implementacja dwóch instrukcji opisanych uprzednio: instrukcji warunkowej i instrukcji cyklu, a następnie implementacja strukturalnej instrukcji wyboru.

Do realizacji instrukcji warunkowej zostaną użyte dwa operatory zdefiniowane w kodzie maszynowym: BRANCH i 0BRANCH.

BRANCH (--)

Wykonanie operacji BRANCH — wchodzącej w skład definicji pewnego operatora — powoduje dodanie do licznika interpretera IP wartości danej słowowej następującej bezpośrednio po wskazaniu operatora BRANCH. Zrealizowanie takiej operacji jest więc równoważne wykonaniu bezwarunkowego skoku względnego.

0BRANCH (*b* --)

Wykonanie operacji 0BRANCH — wchodzącej w skład definicji pewnego operatora — zaczyna się od rozpatrzenia relacji $b = 0$. Jeśli relacja ta jest prawdziwa, to do licznika interpretera IP jest dodawana wartość danej słowowej następującej bezpośrednio po wskazaniu operatora 0BRANCH. W przeciwnym razie, do wspomnianego licznika jest dodawana wartość 2. Zrealizowanie takiej operacji jest więc równoważne wykonaniu warunkowego skoku względnego.

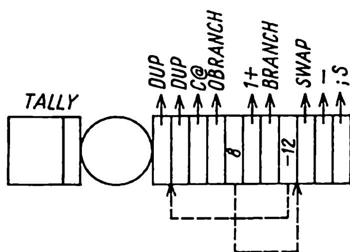
W celu bliższego wyjaśnienia sposobu wykonywania operacji BRANCH i 0BRANCH, zostanie rozpatrzony przykładowy operator TALLY tak dobrany, że wykonanie operacji TALLY powoduje zastąpienie — na stosie parametrów — adresu *a* pola znaków zakończonego znakiem o kodzie 0 daną słowową *n* określającą rozmiar tego pola w znakach. TALLY jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: TALLY
  DUP
  BEGIN
    DUP  C@  WHILE
    1+
  REPEAT
  SWAP  —  ;
```

Poniżej przedstawiono równoważną definicję tego operatora, wyrażoną za pomocą sekwencji zawierającej odwołania do operatorów BRANCH i 0BRANCH

```
: TALLY
  DUP
  DUP C@
  0BRANCH  [  8  ,  ]
  1+
  BRANCH  [  —12  ,  ]
  SWAP  —  ;
```

Strukturę tej definicji pokazano na rys. 8.1.



8.1. Struktura definicji operatora TALLY

Ponieważ podczas interpretowania operacji 0BRANCH licznik interpretera IP wskazuje słowo zawierające daną o wartości 8, wykonanie skoku warunkowego spowoduje nadanie licznikowi interpretera wartości IP + 8. Ponieważ pod adresem IP + 8 znajduje się wskazanie operatora SWAP, ta właśnie operacja zostanie wykonana jako następna po 0BRANCH. Gdyby natomiast skok warunkowy nie miał być wykonany, to do licznika interpretera

zostałaby dodana dana o wartości 2, a więc następną wykonywaną operacją byłoby 1+. Analogicznie można wywnioskować, że podczas wykonywania operacji BRANCH nastąpi przejście do wykonywania drugiej operacji DUP. Pokazano to na rysunku za pomocą linii przerywanych.

Mając do dyspozycji operatory BRANCH i 0BRANCH można przedstawić prostą implementację operatorów kluczowych instrukcji warunkowej IF

```
: IF
  COMPILE 0BRANCH HERE 0 , ; IMMEDIATE
: THEN
  HERE OVER - SWAP ! ; IMMEDIATE
: ELSE
  COMPILE BRANCH HERE 0 , SWAP
  [COMPILE] THEN ; IMMEDIATE
```

W celu ułatwienia analizy przytoczonych definicji, zostanie rozpatrzony przebieg definiowania operatora ?PUT.

```
: ?PUT 0= IF DROP ELSE . THEN ;
```

Operator ten jest zdefiniowany za pomocą kompilatora : (dwukropek), a jego definicja jest równoważna definicji

```
: ?PUT
  0=
  0BRANCH [ 8 , ]
  DROP
  BRANCH [ 4 , ]

;
```

Definiowanie operatora ?PUT przebiega w następujący sposób:

• Zinterpretowanie słowa : powoduje utworzenie w słowniku, nagłówka definicji operatora ?PUT i ustawienie interpretera w stan definiowania.

• Zinterpretowanie słowa 0= powoduje umieszczenie w słowniku wskazania pola kodu operatora 0=.

• Zinterpretowanie słowa IF powoduje wykonanie operacji IF, tj. umieszczenie w słowniku wskazania pola kodu operatora 0BRANCH, a ponadto umieszczenie — na stosie parametrów — adresu pierwszego wolnego bajtu słownika i umieszczenie pod tym adresem danej słowowej o wartości 0.

• Zinterpretowanie słowa DROP powoduje umieszczenie w słowniku wskazania pola kodu operatora DROP.

• Zinterpretowanie słowa ELSE powoduje wykonanie operacji ELSE, tj. umieszczenie w słowniku wskazania pola kodu operatora BRANCH, umieszczenie — na stosie parametrów — adresu pierwszego wolnego bajtu słownika i umieszczenie pod tym adresem danej o wartości 0, a następnie wyznaczenie adresu względnego skoku warunkowego i umieszczenie go

w miejscu danej o wartości 0, występującej po wskazaniu pola kodu operatora 0BRANCH.

• Zinterpretowanie słowa . powoduje umieszczenie w słowniku wskazania pola kodu operatora . (kropka).

• Zinterpretowanie słowa THEN powoduje wykonanie operacji THEN, tj. wyznaczenie adresu względnego skoku bezwarunkowego i umieszczenie go w miejscu danej o wartości 0, występującej po wskazaniu pola kodu operatora BRANCH.

W analogiczny sposób jak operatory kluczowe instrukcji warunkowej mogą być definiowane operatory kluczowe instrukcji strukturalnych. Zostanie to zilustrowane przykładem zdefiniowania instrukcji wyboru, która w ogólnym przypadku będzie wywoływana w kontekście

wyróżnik CASE(
  $cecha_1$ WHEN($ciąg_1$)
  $cecha_2$ WHEN($ciąg_2$)
  . . .
  $cecha_n$ WHEN($ciąg_n$)
  OTHER ($ciąg_0$)
)END

Występujący w tej instrukcji *wyróżnik* oraz wszystkie *cechy* i *ciągi* są dowolnymi sekwencjami operatorów i instrukcji strukturalnych. Wymaga się jedynie, aby zinterpretowanie wyróżnika oraz każdej z cech powodowało umieszczenie — na stosie parametrów — jednej danej słowowej oraz aby zinterpretowanie cechy nie powodowało zmiany rozmiaru stosu powrotów.

Wykonanie instrukcji wyboru polega na porównywaniu danej uzyskanej w następstwie zinterpretowania wyróżnika z danymi uzyskanymi w następstwie zinterpretowania cech. Jeśli dla pewnej $cechy_i$ zostanie stwierdzona równość danych, to nastąpi wykonanie sekwencji $ciąg_i$ i zakończenie wykonywania instrukcji wyboru. Jeśli nie stwierdzi się równości, to nastąpi wykonanie sekwencji $ciąg_0$, chyba że w instrukcji wyboru nie użyto frazy OTHER ($ciąg_0$) i wtedy nie nastąpi wykonanie żadnych czynności.

Posługując się opisaną instrukcją wyboru można zdefiniować przedstawiony uprzednio operator wyboru OBEY. Jak można się przekonać po zinterpretowaniu sekwencji

```
: OBEY
  CASE(
    0 WHEN( DUP )
    1 WHEN( SWAP )
    2 WHEN( DROP )
      OTHER( ." ERROR" )
  )END ;
```

następuje zdefiniowanie takiego operatora OBEY, że

0 OBEY jest równoważne DUP
1 OBEY jest równoważne SWAP
2 OBEY jest równoważne DROP

zaś *n* OBEY dla *n* różnego od 0, 1, 2 jest równoważne wykonaniu operacji powodującej wyprowadzenie napisu ERROR.

Metoda implementowania instrukcji wyboru została przedstawiona na rys. 8.2. Aby ułatwić jej analizę, na rys. 8.3 przedstawiono istotne fragmenty definicji operatora OBEY.

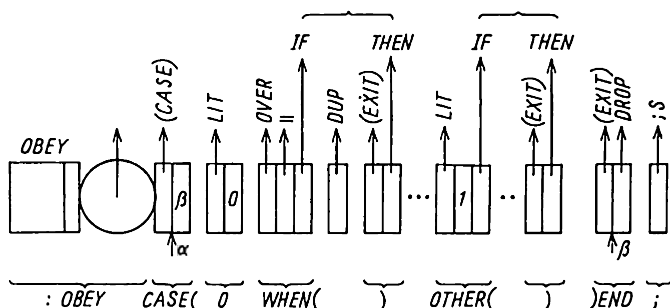
```
( JANB - CASE STATEMENT IMPLEMENTATION )
: (CASE) R> DUP @ >R 2+ >R ;
: (EXIT) R> DROP ;
: CASE( COMPILE (CASE) HERE 0 , ; IMMEDIATE
: WHEN( COMPILE OVER COMPILE =
       [COMPILE] IF ; IMMEDIATE
: OTHER( COMPILE 1 [COMPILE] IF ; IMMEDIATE
: ) COMPILE (EXIT) [COMPILE] THEN ; IMMEDIATE
: )END COMPILE (EXIT) HERE SWAP !
      COMPILE DROP ; IMMEDIATE
```

8.2. Implementacja instrukcji wyboru

Przykład

Implementacja instrukcji cyklu iteracyjnego

```
: (DO)
  R> ROT >R SWAP >R >R ;
: (LOOP)
  R> R> 1+ R> OVER OVER
  < ROT ROT >R >R
  IF
     DUP @ +
  ELSE
     2+
  THEN >R ;
: DO
  COMPILE (DO) HERE ; IMMEDIATE
: LOOP
  COMPILE (LOOP) HERE - , ; IMMEDIATE
```



8.3. Struktura definicji operatora OBEY

• W takiej implementacji, zmienna sterująca cyklu może być odzyskana za pomocą operacji I

```
: I
R> R> DUP >R SWAP >R ;
```

• W praktyce operacje (DO), (LOOP) i I są zdefiniowane w kodzie maszynowym. Powoduje to znaczne przyśpieszenie wykonywania cyklu. □

9. Słownik operatorów

Słownik operatorów języka Forth jest *strukturą drzewiastą* podzieloną na *podsłowniki*. Słownik operatorów składa się z co najmniej jednego podsłownika o nazwie FORTH. W podsłowniku tym znajdują się definicje podstawowych operatorów języka, w tym operatorów kluczowych do definiowania instrukcji strukturalnych. Najczęściej używanymi podsłownikami dodatkowymi są EDITOR i ASSEMBLER. W pierwszym z nich znajdują się definicje operatorów umożliwiających przygotowywanie i redagowanie programów źródłowych. W drugim natomiast znajdują się definicje operatorów do generowania rozkazów w kodzie maszynowym.

Do zarządzania słownikami i podsłownikami służą operatory VOCABULARY, DICTIONARY, DEFINITIONS, CONTEXT i CURRENT. Wykonanie sekwencji

VOCABULARY *nazwa* IMMEDIATE

powoduje utworzenie podsłownika *nazwa* i związanie go z bieżącym słownikiem definiowania operatorów. Bezpośrednio po zainicjowaniu procesora podsłownikiem tym jest FORTH.

Jeśli po wykonaniu tej czynności zostanie wykonana operacja *nazwa*, to bieżącym podsłownikiem wyszukiwania operatorów stanie się podsłownik *nazwa*. Fakt ten zostanie odnotowany w zmiennej identyfikowanej przez operator CONTEXT. Aż do następnej zmiany podsłownika wyszukiwania, przeglądanie słownika w celu znalezienia definicji pewnego operatora będzie dotyczyć w pierwszej kolejności podsłownika *nazwa*, a następnie podsłownika, w którym została umieszczona definicja operatora *nazwa*, tj. — w najprostszym przypadku — podsłownika FORTH.

Zmiana bieżącego podsłownika definiowania może być zrealizowana za pomocą operatora DEFINITIONS użytego w kontekście

nazwa DEFINITIONS

gdzie *nazwa* jest operatorem zdefiniowanym za pomocą operatora VOCABULARY.

Wykonanie rozpatrywanej sekwencji powoduje taką zmianę wartości zmiennej identyfikowanej przez operator CURRENT, że bieżącym podsłownikiem definiowania staje się ten sam podsłownik, który jest bieżącym słownikiem wyszukiwania. Oznacza to np., że jeśli w pewnym momencie zostanie wykonana sekwencja

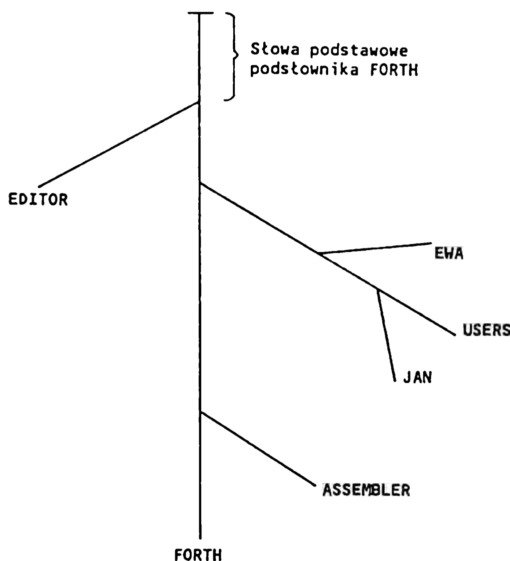
VOCABULARY JOHN

a następnie zostanie wykonana sekwencja

JOHN DEFINITIONS

to zarówno bieżącym podsłownikiem definiowania jak i bieżącym podsłownikiem wyszukiwania będzie JOHN.

W ogólnym przypadku struktura słownika może być dość złożona i może mieć np. postać przedstawioną na rys. 9.1. W tej przykładowej



9.1. Struktura słownika operatorów

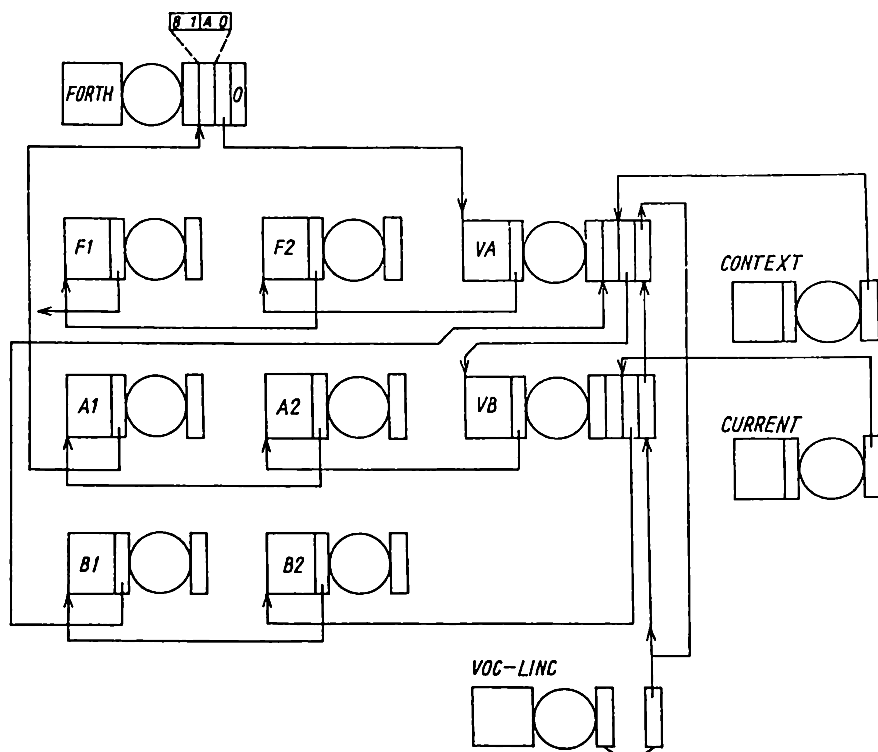
strukturze w podsłowniku FORTH znajdują się definicje podsłowników EDITOR, USERS i ASSEMBLER, a w podsłowniku USERS znajdują się definicje podsłowników JAN i EWA. Utworzenie takiej struktury słownika można by np. uzyskać za pomocą sekwencji

VOCABULARY EDITOR IMMEDIATE
VOCABULARY USERS IMMEDIATE
USERS DEFINITIONS

VOCABULARY EWA IMMEDIATE
VOCABULARY JAN IMMEDIATE
FORTH DEFINITIONS
VOCABULARY ASSEMBLER IMMEDIATE

Nic nie stoi na przeszkodzie, aby bieżący podsłownik wyszukiwania operatorów był różny od bieżącego podsłownika definiowania. W szczególności, jeśli bieżącym podsłownikiem definiowania jest JAN, a bieżącym podsłownikiem wyszukiwania jest EWA, to nowe definicje operatorów są umieszczane w podsłowniku JAN, a wyszukiwanie operatorów odbywa się w całym podsłowniku EWA, w całym podsłowniku USERS i w całym podsłowniku FORTH. Oczywiście wyszukiwanie kończy się natychmiast po znalezieniu operatora. Dzięki temu, w rozłącznych słownikach, mogą występować definicje operatorów o tej samej nazwie.

W celu bliższego zilustrowania powiązań występujących między operatorami zdefiniowanymi w słowniku na rys. 9.2 przedstawiono skutek zinter-



9.2. Powiązania zmiennych, podsłowników i operatorów

pretowania — bezpośrednio po zainicjowaniu procesora — sekwencji

```
: F1 ;
: F2 ;
VOCABULARY VA IMMEDIATE
VA DEFINITIONS
: A1 ;
: A2 ;
VOCABULARY VB IMMEDIATE
VB DEFINITIONS
: B1 ;
: B2 ;
VA
```

Jak można się przekonać, w słowniku występują listy łączące podsłowniki, jak również listy łączące operatory należące do poszczególnych podsłowników. W rozpatrywanym przykładzie wartości zmiennych identyfikowanych przez operatory CONTEXT i CURRENT są różne, ponieważ po wykonaniu przytoczonej sekwencji bieżącym podsłownikiem wyszukiwania jest VA, a bieżącym podsłownikiem definiowania jest VB.

Bliższa analiza definicji operatorów identyfikujących podsłowniki, takich jak np. VA i VB wykazuje, że w polu parametrów każdego z nich znajduje się definicja pseudooperatora, którego nazwą jest spacja. Oczywiście, definicja taka może być wytworzona jedynie sztucznie, gdyż spacja nie jest słowem. Jest ona jedynie separatorem słów strumienia wejściowego.

10. Edytor i asembler

Jak już wspomniano, do najczęściej definiowanych słowników pomocniczych należą EDITOR i ASSEMBLER. Oba te podsłowniki znajdują się poza definicją języka fig-Forth i tym samym mogą być realizowane na wiele sposobów. W każdym z nich znajduje się ustalona przez implementację liczba definicji operatorów.

Operatory podsłownika EDITOR umożliwiają przetwarzanie znaków zapisanych w ekranach, natomiast operatory podsłownika ASSEMBLER umożliwiają definiowanie operatorów zawierających odwołania do mnemonik rozkazów maszynowych i rejestrów.

Podsłownik EDITOR

Podczas edycji ekranu, operatory edytora odwołują się do obszaru pamięci operacyjnej nazywanego dalej *buforem tekstu.* W buforze tym mieści się jeden wiersz tekstu. Ekran, który ma być poddany edycji, jest sprowadzany do pamięci operacyjnej w następstwie zinterpretowania operacji LIST. Po wykonaniu edycji pewnej porcji ekranów, które w nowej postaci znajdują się na ogół w pamięci operacyjnej, należy wykonać operację FLUSH, która powoduje przeniesienie zaktualizowanych ekranów do pamięci zewnętrznej. Typowe edytory umożliwiają ponadto zarówno wykonywanie operacji na wierszach, jak również wykonywanie operacji na zawartych w nich ciągach znaków.

Przygotowanie procesora do posługiwania się definicjami podsłownika EDITOR wymaga wykonania operacji *n* LOAD, gdzie *n* jest numerem ekranu. Spowodowane tym zinterpretowanie definicji zawartych w ekranie *n* i ewentualnie ekranach następnych (jeśli posłużono się operatorami — — >) spowoduje m.in. utworzenie definicji podsłownika EDITOR i umieszczenie w nim definicji nowych operatorów.

Dokładny zestaw i postać operatorów edycyjnych zależy od implementacji. Z tego powodu zostaną przytoczone opisy przykładowe.

PAD (— — *a*) (zdefiniowany w podsłowniku FORTH)
Wykonanie operacji PAD powoduje umieszczenie na szczycie stosu parametrów adresu *a* bufora tekstu.

LIST (*n* — —) (zdefiniowany w podsłowniku FORTH)
Wykonanie operacji LIST powoduje sprowadzenie do pamięci operacyjnej ekranu o numerze *n* i przygotowanie go do edycji.

CLEAR (*n* — —)
Wykonanie operacji CLEAR powoduje zapełnienie ekranu o numerze *n* spacjami i przygotowanie go do edycji.

COPY (*n*1 *n*2 — —)
Wykonanie operacji COPY powoduje przeniesienie zawartości ekranu o numerze *n*1 do ekranu o numerze *n*2.

FLUSH (— —)
Wykonanie operacji FLUSH powoduje przeniesienie do pamięci zewnętrznej ekranów zaktualizowanych w pamięci operacyjnej.

H (*n* — —)
Wykonanie operacji H (*hold*) powoduje skopiowanie wiersza o numerze *n* do bufora tekstu.

D (*n* — —)
Wykonanie operacji D (*delete*) powoduje skopiowanie wiersza o numerze *n* do bufora tekstu, a następnie usunięcie tego wiersza. Powoduje to przesunięcie pozostałych wierszy "w górę" i zapełnienie wiersza nr 15 spacjami.

T (*n* — —)
Wykonanie operacji T (*type*) powoduje umieszczenie wiersza o numerze *n* w buforze tekstu, a następnie wyprowadzenie go. Kursor zostanie usytuowany na początku wiersza *n*.

R (*n* — —)
Wykonanie operacji R (*replace*) powoduje zastąpienie wiersza o numerze *n* zawartością bufora tekstu.

I (*n* — —)
Wykonanie operacji I (*insert*) powoduje wstawienie zawartości bufora tekstu w miejsce wiersza o numerze *n* oraz przesunięcie tego wiersza i wierszy następnych "w dół" o jeden wiersz (wiersz nr 15 "gubi się").

E (*n* --)

Wykonanie operacji E (*erase*) powoduje zastąpienie wiersza o numerze *n* wierszem składającym się z samych spacji.

S (*n* --)

Wykonanie operacji S (*spread*) powoduje przesunięcie wiersza o numerze *n* oraz wierszy następnych o jeden wiersz "w dół", a następnie umieszczenie w wierszu o numerze *n* samych spacji (wiersz nr 15 "gubi się").

L (--)

Wykonanie operacji L (*list*) powoduje wyprowadzenie zawartości bieżącego ekranu.

TOP (--)

Wykonanie operacji TOP powoduje usytuowanie kursora na początku bieżącego ekranu.

M (*n* --)

Wykonanie operacji M (*move*) powoduje przesunięcie kursora o *n* pozycji (*n* może być ujemne).

F (--)

Wykonanie operacji F (*find*) powoduje wprowadzenie ze strumienia wejściowego tekstu zakończonego znakiem cr, a następnie odszukanie tego tekstu na ekranie. Poszukiwanie rozpoczyna się od bieżącej pozycji kursora. Jeśli tekst nie zostanie znaleziony, to jest wyprowadzany komunikat, a kursor jest ustawiany tak jak po wykonaniu operacji TOP. Jeśli tekst zostanie znaleziony, to kursor zostanie ustawiony w pozycji za tym tekstem.

N (--)

Wykonanie operacji N (*next*) powoduje znalezienie następnego wystąpienia słowa znalezionego za pomocą operacji F.

X (--)

Wykonanie operacji X (*cross*) powoduje wprowadzenie ze strumienia wejściowego tekstu zakończonego znakiem cr, a następnie odszukanie tego tekstu na ekranie i usunięcie go.

C (--)

Wykonanie operacji C (*copy*) powoduje wprowadzenie ze strumienia wejściowego tekstu zakończonego znakiem cr, a następnie umieszczenie go na ekranie, począwszy od bieżącej pozycji kursora.

B (−−)

Wykonanie operacji B (*back*) powoduje przesunięcie kursora wstecz o taką liczbę pozycji, ile znaków liczył ostatni tekst stanowiący argument operacji F, X albo C.

TILL (−−)

Wykonanie operacji TILL (*till end*) powoduje wprowadzenie ze strumienia wejściowego tekstu zakończonego znakiem cr, a następnie usunięcie z wiersza, w którym znajduje się kursor, począwszy od bieżącej pozycji kursora, grupy znaków zakończonej wprowadzonym tekstem.

Przykład

Wykorzystanie operatorów podsłownika EDITOR

Jeśli w wierszu nr 1 ekranu o numerze 200 znajduje się definicja

: SET COMPILE GET ;

to zmiana pierwszego operatora COMPILE na [COMPILE] może być zrealizowana np. za pomocą sekwencji

```
EDITOR
200 LIST 1 T
X COMPILE
C [COMPILE]
FLUSH
```

□

Podsłownik ASSEMBLER

Zestaw operatorów zdefiniowanych w podsłowniku ASSEMBLER zależy od implementacji procesora. Do zestawu tego należą przede wszystkim operatory o nazwach identycznych z mnemonikami rozkazów i rejestrów maszynowych, a ponadto operatory kluczowe umożliwiające posługiwanie się asemblerowymi instrukcjami strukturalnymi. Ewentualne kolizje nazw operatorów występujących w podsłowniku ASSEMBLER i nazw operatorów innych podsłowników rozstrzygane są wg zasad wyszukiwania operatorów w słowniku. Z tego względu nie kolidują ze sobą np. operatory kluczowe IF występujące zarówno w podsłowniku FORTH jak i w podsłowniku ASSEMBLER.

Przygotowanie procesora do posługiwania się definicjami podsłownika ASSEMBLER wymaga wykonania operacji *n* LOAD, gdzie *n* jest numerem ekranu. Spowodowane tym zinterpretowanie definicji zawartych w ekranie *n* i ewentualnie ekranach następnych (jeśli posłużono się operatorami −−>) spowoduje m.in. utworzenie definicji podsłownika ASSEMBLER, umieszczenie

w nim definicji nowych operatorów, a ponadto rozbudowanie podsłownika FORTH o definicje operatorów CODE i C; i modyfikację operatora ;CODE.

Dokładna postać operatorów zdefiniowanych w podsłowniku ASSEMBLER, jak również postać operatorów CODE i C; zależy od implementacji. Z tego powodu zostaną przytoczone opisy przykładowe

CODE (-- *b c*)

Wykonanie operacji CODE zaczyna się od upewnienia, iż interpreter znajduje się w stanie wykonywania. Następnie na stosie parametrów jest umieszczana dana słowowa *b*, której wartość określa bieżącą podstawę liczb, oraz dana słowowa *c*, mająca wartość zmiennej identyfikowanej przez operator CONTEXT. Po wykonaniu tych czynności zostaje utworzony nagłówek definicji nowego operatora o nazwie wyrażonej przez słowo występujące po operatorze CODE. Na zakończenie, w zmiennej identyfikowanej przez operator CSP jest zapamiętywany bieżący rozmiar stosu parametrów i wykonywana jest operacja ASSEMBLER. Wykonanie tej operacji powoduje, że bieżącym podsłownikiem wyszukiwania operatorów staje się ASSEMBLER. W nim właśnie znajdują się definicje operatorów o nazwach identycznych z mnemonikami rozkazów maszynowych i rejestrów. CODE jest operatorem biernym, zdefiniowanym za pomocą kompilatora: (dwukropek)

```
: CODE  ?EXEC  BASE  @  CONTEXT  @
        CREATE !CSP [COMPILE] ASSEMBLER ;
```

C; (*c b* --)

Wykonanie operacji C; zaczyna się od umieszczenia w słowniku rozkazu skoku do interpretera wewnętrznego. Rozkaz ten staje się ostatnim w ciągu rozkazów definiujących operator w kodzie maszynowym. Następnie upewnia się, że podczas definiowania operatora nie uległ zmianie rozmiar stosu parametrów. Po wykonaniu tych czynności przywraca się słownik wyszukiwania i podstawę liczb zapamiętane na stosie parametrów podczas wykonywania operacji CODE. Na zakończenie neguje się bit *smudge* występujący w nagłówku właśnie definiowanego operatora. C; jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: C;  NEXT1  C3  C,  ,  ?CSP
      CONTEXT  !  BASE  !  SMUDGE  ;
```

NEXT1 i C3 wyrażają odpowiednio adres i część operacyjną wspomnianego rozkazu skoku. Ponieważ dane te zależą od implementacji, posłużono się szesnastkowym kodem rozkazu skoku dla mikroproce-

sorów Intel 8080 i Z80. W implementacjach języka fig-Forth na tych mikroprocesorach z reguły obowiązuje następująca definicja adresu NEXT1

' DROP 2+ @ CONSTANT NEXT1

Posługiwanie się operatorami CODE i C; do definiowania nowych operatorów za pomocą sekwencji·

CODE *nazwa rozkazy* C;

wymaga znajomości listy rozkazów komputera, w którym implementowano język Forth. Równie ważna jest znajomość struktury interpretera wewnętrznego.

Dla skonkretyzowania dalszego opisu, przytoczone tu wyjaśnienia zostaną ograniczone do przypadku listy rozkazów mikroprocesora Intel 8080 oraz interpretera wewnętrznego przedstawionego na rys. 10.1. W celu ułatwienia posługiwania się rozkazami maszynowymi, w Dodatku B zamieszczono krótki opis mikroprocesora Intel 8080 oraz wyszczególniono listę jego rozkazów.

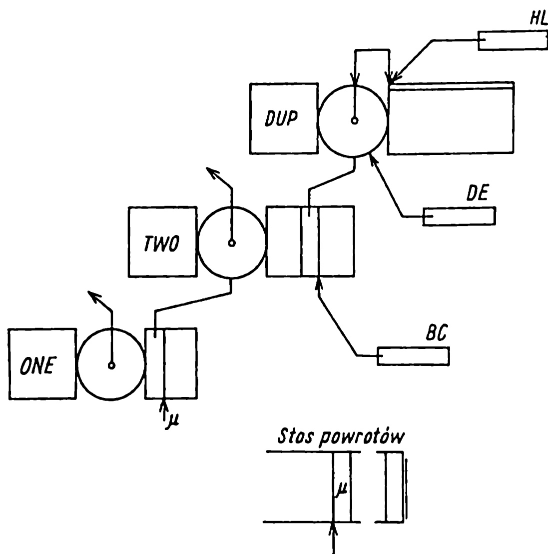
```
pushDE : PUSH D
pushHL : PUSH H
NEXT1  : LDAX B
         MOV  L,A
         INX  B
         LDAX B
         MOV  H,A
         INX  B
NEXT2  : MOV  E,M
         INX  H
         MOV  D,M
         XCHG
         PCHL
```

10.1. Interpreter wewnętrzny

Podczas wykonywania interpretacji, rolę licznika interpretera IP odgrywa rejestr BC. W rejestrze tym jest przechowywany adres wskazania następnego interpretowanego operatora lub adres wskazania danej stanowiącej słownikowy argument właśnie interpretowanego operatora. W chwili podjęcia wykonywania operacji wskazanej przez pole kodu bieżąco interpretowanego operatora rejestr DE zawiera wskazanie drugiego bajtu tego pola kodu, a rejestr HL zawiera adres podprogramu określonego przez to pole. Spośród trzech wymienionych tu rejestrów, jedynie BC musi być zachowany między kolejnymi wykonaniami operatorów. W przypadku gdy bieżący operator nie jest zdefiniowany za pomocą kodu, lecz za pomocą innych operatorów,

wymaga to okresowego przechowywania tego rejestru na stosie powrotów, tak jak ilustrowano to już uprzednio.

W celu poglądowego przedstawienia właściwości analizowanej tu implementacji języka Forth na rys. 10.2 uwidoczniono stan rejestrów BC, DE



10.2. Rejestry interpretera wewnętrznego

i HL oraz stan stosu powrotów, w momencie podjęcia wykonywania operacji DUP – wyrażonej w kodzie maszynowym i wywoływanej podczas realizowania operacji TWO, wywołanej z kolei podczas wykonywania operacji ONE.

Przykład

Zdefiniowanie operatora SWITCH, którego zadaniem jest zamiana miejscami górnego i dolnego bajtu danej słowowej znajdującej się na szczycie stosu parametrów.

```
CODE SWITCH
     H POP      ( POP H )
     A L MOV    ( MOV A,L )
     L H MOV    ( MOV L,H )
     H A MOV    ( MOV H,A )
     H PUSH     ( PUSH H )
     C;
```

• Pole parametrów przytoczonej definicji składa się z 8 bajtów.

• Zdefiniowanie tego samego operatora za pomocą kompilatora : (dwukropek)

```
: SWITCH
DUP  SP@ 1+ @ SWAP
DROP SWAP DROP ;
```

spowodowałoby utworzenie pola parametrów składającego się z 16 bajtów.

• Czas wykonywania operacji SWITCH wyrażonej za pomocą operatorów CODE i C; jest 10-krotnie krótszy od czasu wykonania tej operacji wyrażonej za pomocą operatorów : i ;.

• Wadą definiowania za pomocą operatorów CODE i C; jest konieczność posługiwania się rozkazami maszynowymi w notacji odwrotnej (np. H POP zamiast POP H) oraz wymaganie znajomości implementacji interpretera wewnętrznego. W wielu przypadkach niezbędna jest także znajomość adresów niektórych zmiennych procesora; np. zmiennej określającej bieżący adres szczytu stosu powrotów. □

11. Definiowanie asemblerów

Posługiwanie się podsłownikiem ASSEMBLER, z jego definicjami mnemonik rozkazowych, nie stanowi warunku koniecznego definiowania operatorów w kodzie maszynowym.

Jeśli używanym mikroprocesorem jest INTEL 8080, a struktura interpretera wewnętrznego jest taka jak podano w poprzednim rozdziale, to definicje operatorów wyrażone za pomocą rozkazów maszynowych mogą być przedstawione za pomocą dwóch pomocniczych operatorów nazwanych tu :: i ;;.

:: (-- *b*)

Wykonanie operacji :: powoduje wprowadzenie ze strumienia wejściowego jednego słowa i utworzenie definicji operatora o nazwie wyrażonej przez to słowo. W polu kodu zostaje umieszczony adres pola parametrów, w polu nazwy zostaje ustawiony bit *smudge*, a na stosie parametrów zostaje zapamiętana bieżąca podstawa liczb. Po wykonaniu tych czynności podstawa liczb zostaje zmieniona na szesnastkową, a interpreter zostaje ustawiony w stan wykonywania. :: zdefiniowano tu jako operator bierny, posługując się kompilatorem : (dwukropek)

```
: :: CREATE BASE @ HEX ;
```

;; (*b* --)

Wykonanie operacji ;; powoduje umieszczenie w słowniku rozkazu JMP NEXT reaktywującego interpreter, usunięcie bitu *smudge* ustawionego w polu nazwy podczas wykonywania operacji :: oraz odtworzenie ze stosu parametrów pierwotnej podstawy liczb. ;; zdefiniowano tu jako operator bierny, posługując się kompilatorem : (dwukropek)

```
: ;; NEXT1 C3 C, ,
     SMUDGE BASE ! ;
```

Dysponując takimi definicjami — stanowiącymi rozszerzenie języka Forth — można podać alternatywną definicję rozpatrywanego uprzednio operatora SWITCH

```
:: SWITCH
   E1 C, ( POP H )
   7D C, ( MOV A,L )
   6C C, ( MOV L,H )
   67 C, ( MOV H,A )
   E5 C, ( PUSH H )
   ;;
```

Oczywiście takie postępowanie nie jest zbyt wygodne, ponieważ wymaga posługiwania się rozkazami w ich reprezentacji szesnastkowej. Tym niemniej stanowi ono potwierdzenie tezy, iż tworzenie definicji wyrażonych w kodzie maszynowym nie wymaga posługiwania się operatorami podsłownika ASSEMBLER.

W analogiczny sposób, a więc bez posługiwania się mnemonikami rozkazów i rejestrów, mogą być definiowane kompilatory. Przykładem tego może być definicja kompilatora VECTOR, której można nadać postać

```
: VECTOR CREATE   SMUDGE  ALLOT
         ;CODE    BASE @  HEX
            E1    C,             ( POP H )
            13    C,             ( INX D )
            19    C,             ( DAD D )
            E5    C,             ( PUSH H )
            NEXT1 C3 C, ,        ( JMP NEXT1 )
         BASE !
```

Zinterpretowanie przytoczonej sekwencji operatorów powoduje utworzenie definicji operatora VECTOR, który ma dokładnie takie same właściwości jak operator VECTOR zdefiniowany za pomocą operatorów <BUILDS i DOES>

```
: VECTOR <BUILDS ALLOT DOES> + ;
```

jednak oszczędniej gospodaruje pamięcią operacyjną, ponieważ kompiluje operatory, których pola parametrów są o 2 bajty krótsze.

Występująca w "oszczędniejszej" wersji kompilatora operacja ;CODE jest operacją czynną, której wykonanie powoduje m.in. umieszczenie w słowniku wskazania operatora (;CODE), wykonanie operacji SMUDGE dotyczącej kompilatora VECTOR oraz ustawienie interpretera w stan wykonywania. Podczas przyszłego wykonania operacji VECTOR, np. w kontekście

```
5 VECTOR STRING
```

wykonanie operacji (;CODE) spowoduje taką zmianę pola kodu operatora STRING, że będzie ono zawierać adres części wykonawczej kompilatora, tj. tego bajtu słownika, w którym umieszczono rozkaz POP H.

Należy zwrócić uwagę na to, iż użycie operatora SMUDGE w definicji kompilatora VECTOR jest niezbędne. Gdyby nie użyto tego operatora, to operator STRING nie byłby w pełni zdefiniowany, a zatem nie mógłby być odnaleziony w słowniku.

Biorąc pod uwagę to, iż definiowanie operatorów i kompilatorów bez odwoływania się do mnemonik nie jest zbyt wygodne, ponieważ wymaga posługiwania się rozkazami w ich reprezentacji szesnastkowej, w większości implementacji języka Forth występuje zbiór ekranów zawierających definicje mnemonik podsłownika ASSEMBLER.

W celu wyjaśnienia zasad przygotowywania treści wymienionych ekranów na rys. 11.1 przytoczono niezbędne definicje umożliwiające posługiwanie się mnemonikami rozkazowymi. Ponadto podano tam także kilka przykładowych operatorów kluczowych umożliwiających asemblerowe programowanie strukturalne.

Sposób posługiwania się przytoczonymi definicjami zostanie wyjaśniony na przykładzie operatora ?SWAP. Operator ten jest zdefiniowany w taki sposób, że wykonanie operacji ?SWAP powoduje zdjęcie ze stosu parametrów i rozpatrzenie danej słowowej, a następnie — jeśli ma ona wartość różną od 0 — wykonanie operacji SWAP.

Operator ?SWAP mógłby zostać zdefiniowany za pomocą kompilatora : (dwukropek) jako

```
: ?SWAP  IF  SWAP  THEN  ;
```

ale w celu zilustrowania zasad posługiwania się operatorami podsłownika ASSEMBLER zostanie zdefiniowany za pomocą operatora CODE

```
CODE ?SWAP
      H POP
      A L MOV
      H ORA
      0= NOT
      IF
        H POP
        XTHL
        H PUSH
      THEN
      C;
```

Przebieg definiowania operatora ?SWAP wyrażonego za pomocą sekwencji odwołań do operatorów podsłownika ASSEMBLER jest następujący:

- Wykonanie operacji CODE powoduje m.in. utworzenie nagłówka definicji operatora ?SWAP i uczynienie podsłownika ASSEMBLER bieżącym podsłownikiem wyszukiwania.
- Wykonanie operacji H powoduje umieszczenie — na stosie parametrów — danej słowowej o wartości 4, identyfikującej rejestr H.

```
FORTH  DEFINITIONS
VOCABULARY  ASSEMBLER  IMMEDIATE
: CODE  BASE  @  CONTEXT  @  CREATE
        !CSP  [COMPILE]  ASSEMBLER  ;
ASSEMBLER  DEFINITIONS
HEX
   DROP  2+  @  CONSTANT  NEXT1
: C;  NEXT1  C3  C,  ,  ?CSP
      CONTEXT  !  BASE  !  SMUDGE  ;
: ;C  NEXT1  C3  C,  ,  ?CSP
      CURRENT  @  CONTEXT  !  SMUDGE  ;
: 8*   DUP  +  DUP  +  DUP  +  ;
: 1MI  <BUILDS  C,  DOES>  C@  C,  ;
: 2MI  <BUILDS  C,  DOES>  C@  +  C,  ;
: 3MI  <BUILDS  C,  DOES>  C@  SWAP  8*  +  C,  ;
: 4MI  <BUILDS  C,  DOES>  C@  C,  ,  ;
0  CONSTANT  B
1  CONSTANT  C
2  CONSTANT  D
3  CONSTANT  E
4  CONSTANT  H
5  CONSTANT  L
6  CONSTANT  M
7  CONSTANT  A
   E3  1MI  XTHL
   37  1MI  STC
   1F  1MI  RAR
   B0  2MI  ORA
   C1  3MI  POP
   C5  3MI  PUSH
   03  3MI  INX
   C3  4MI  JMP
   CD  4MI  CALL
: MOV  SWAP  8*  40  +  +  C,  ;
C2  CONSTANT  0=      ( jnz )
F2  CONSTANT  0<      ( jp  )
: NOT   8  OR  ;
: IF    C,  HERE  0  ,  ;
: THEN  HERE  SWAP  !  ;
: ELSE  C3  IF  ROT  SWAP  THEN  ;
'  ASSEMBLER  CFA  '  ;CODE  8  +  !
DECIMAL
FORTH  DEFINITIONS
```

11.1. Definicje operatorów podsłownika ASSEMBLER

• Wykonanie operacji POP, skompilowanej za pomocą kompilatora 3MI, powoduje wykonanie sekwencji

C@ SWAP 8* + C,

tj. umieszczenie na stosie parametrów, a następnie przeniesienie do słownika, danej bajtowej reprezentującej rozkaz POP H.

• Wykonanie sekwencji operacji

A L MOV

powoduje w analogiczny sposób umieszczenie w słowniku bajtu, reprezentującego rozkaz MOV A,L.

• Wykonanie sekwencji operacji

0= NOT

powoduje umieszczenie na stosie parametrów części operacyjnej rozkazu JZ.

• Wykonanie operacji

IF

powoduje umieszczenie w słowniku rozkazu JZ *a*, gdzie *a* jest adresem ostatniego bajtu słownika tuż po wykonaniu operacji THEN.

• Wykonanie operacji C; powoduje umieszczenie w definicji operatora ?SWAP rozkazu maszynowego JMP NEXT1 oraz usunięcie z nagłówka definiowanego operatora bitu *smudge*, umieszczonego tam podczas wykonywania operacji CODE.

• Ostatecznie pole parametrów operatora ?SWAP uzyskuje postać ciągu rozkazów, który można przedstawić symbolicznie jako

```
    POP   H
    MOV   A,L
    ORA   H
    JZ    skip
    POP   H
    XTHL
    PUSH  H
skip:
    JMP   NEXT1
```

Poprzestając na tych wyjaśnieniach, pozostaje jedynie uzasadnić, jakiemu celowi służy sekwencja

' ASSEMBLER CFA ' ;CODE 8 + !

występująca w ostatnim wierszu rys. 11.3.

Przed wykonaniem tej sekwencji definicja operatora ;CODE ma postać

```
: ;CODE ?CSP COMPILE (;CODE)
        [COMPILE] [ SMUDGE ; IMMEDIATE
```

a po jej wykonaniu ma postać

```
: ;CODE ?CSP COMPILE (;CODE)
        [COMPILE] [ [COMPILE] ASSEMBLER
        ; IMMEDIATE
```

Dzięki dokonaniu takiej zmiany, bezpośrednio po operatorze ;CODE mogą występować mnemoniki rozkazów maszynowych – zakończone odwołaniem do operatora ;C. Umożliwia to zapisanie rozpatrywanej uprzednio definicji kompilatora VECTOR w postaci znacznie wygodniejszej, a mianowicie

```
: VECTOR  CREATE  SMUDGE  ALLOT
          ;CODE
             H POP
             D INX
             D DAD
             H PUSH
          ;C
```

12. Programowanie hybrydowe Forth-asembler

Posługiwanie się operatorami CODE, ;CODE (w wersji zmodyfikowanej) oraz C; umożliwia łatwe definiowanie za pomocą rozkazów maszynowych prostych operatorów i części wykonawczych kompilatorów.

Chociaż posługiwanie się rozkazami w kodzie maszynowym powinno być zredukowane do minimum, niejednokrotnie okazuje się, że repertuar środków programowych dostarczonych przez język Forth jest zbyt ubogi. W takich przypadkach nieodzowne jest odwołanie się do programowania asemblerowego.

Jeśli potrzeba odwołania się do operatora wyrażonego za pomocą rozkazów maszynowych ujawnia się w ciągu operatorów języka Forth, to wystarczy uprzednio zdefiniować pomocniczy operator o definicji wyrażonej w kodzie maszynowym i odwołać się do niego. Sytuacja jest nieco trudniejsza, gdy pożądane jest, aby w ciągu rozkazów maszynowych wystąpiły odwołania do operatorów spoza podsłownika ASSEMBLER. W takim przypadku należy definicję rozbić na kilka prostszych i pomiędzy odwołania do nich wstawić odwołania do operatorów spoza wspomnianego podsłownika.

Alternatywnym rozwiązaniem problemu *programowania hybrydowego*, tj. łączenia instrukcji języka wysokiego poziomu i asemblera, jest opracowanie definicji operatorów hybrydowych umożliwiających dowolne przełączanie się między operatorami zwykłymi i rozkazowymi.

Chociaż trudno jest ocenić zakres zastosowań takich operatorów, nazwanych tu ASM(i)FORTH oraz FORTH(i)ASM, przeanalizowanie ich definicji należy uznać za pożyteczny sprawdzian biegłości posługiwania się językiem Forth.

Przykładowe definicje hybrydowe zawierające odwołania do wymienionych tu operatorów przedstawiono na rys. 12.1 i 12.2. Jak można zauważyć,

```
: 2/  DUP  @
    ASM(  H POP
          A H MOV
          A ORA
          0<  IF
            STC
          THEN
          RAR
          H A MOV
          A L MOV
          RAR
          L A MOV
          H PUSH  )FORTH
    SWAP  !  ;
```

12.1. Definicja operatora 2/

definicja operatora 2/ rozpoczyna się od operatorów zwykłych, po czym następuje operator ASM(, ciąg operatorów rozkazowych, operator)FORTH i ponownie operatory zwykłe. Definicja operatora 2* rozpoczyna się od ciągu operatorów rozkazowych, po którym następuje operator FORTH(, operatory zwykłe, operator)ASM i ponownie operatory rozkazowe.

```
CODE 2*  H POP
         H PUSH
         H PUSH
         FORTH(
           @  DUP  +
         )ASM
         D POP
         H POP
         M E MOV
         H INX
         M D MOV
         C;
```

12.2. Definicja operatora 2*

Wykonanie operacji 2/ i 2* powoduje odpowiednio: podzielenie i pomnożenie przez 2 danej słowowej, której adres znajduje się na szczycie stosu parametrów. Między innymi powoduje to, że w następstwie wykonania sekwencji −81 SP@ 2/ zostanie wyprowadzona liczba −41, a w następstwie wykonania sekwencji 40 SP@ 2* zostanie wyprowadzona liczba 80.

```
FORTH DEFINITIONS
: ASM( BASE @ CONTEXT @ HERE 6 + ,
       COMPILE BRANCH HERE 0 ,
       HERE 2+ , [COMPILE] ASSEMBLER
       [COMPILE] [ ; IMMEDIATE

ASSEMBLER DEFINITIONS
: )FORTH NEXT JMP HERE OVER - SWAP !
         CONTEXT ! BASE ! ] ;

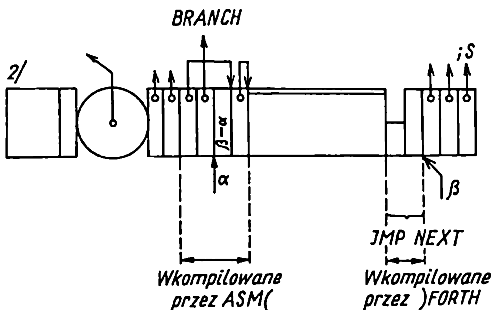
ASSEMBLER DEFINITIONS
CODE (POP) L C MOV
           H B MOV
           XTHL
           C L MOV
           B H MOV
           C;
: FORTH( BASE @ CONTEXT @ ' (POP)
         CALL ] COMPILE >R ;

FORTH DEFINITIONS
: )ASM COMPILE R> HERE 2+ , HERE 2+ ,
       ASSEMBLER B POP FORTH
       CONTEXT ! BASE ! [COMPILE] ASSEMBLER
       [COMPILE] [ ; IMMEDIATE
```

12.3. Definicje operatorów do programowania hybrydowego

Proponowaną implementację czterech operatorów hybrydowych przedstawiono na rys. 12.3. Dla ułatwienia ich analizy na rys. 12.4 przytoczono strukturę definicji operatora 2/, a na rys. 12.5 wybrane fragmenty struktur definicji operatorów

FORTH(,)ASM, (POP) i 2*.



12.4. Struktura definicji operatora 2/

12.5. Struktury definicji operatorów do programowania hybrydowego

13. Komunikowanie się z terminalem

Terminalem, z którym komunikuje się procesor języka Forth, jest zazwyczaj konsola składająca się z klawiatury i monitora albo klawiatury i drukarki. Wszystkie operacje wejścia/wyjścia, które dotyczą terminala wykorzystują operatory podstawowe KEY, EMIT, CR i ?TERMINAL. Wykonanie operacji C/L powoduje umieszczenie — na szczycie stosu parametrów — danej słowowej określającej rozmiar wiersza ekranu w znakach.

Wprowadzenie większych porcji znaków umożliwiają operatory EXPECT, WORD i QUERY. Po wprowadzeniu danych znakowych do pamięci operacyjnej mogą być one przetworzone na dane liczbowe. Do tego celu służą operatory NUMBER i DPL. Podczas wykonywania takiej konwersji jest uwzględniana bieżąca podstawa liczb, określona przez zmienną identyfikowaną przez operator BASE.

EXPECT (*a n* --)

Wykonanie operacji EXPECT powoduje wprowadzenie z klawiatury terminala nie więcej niż *n* znaków i umieszczenie ich w polu pamięci operacyjnej, począwszy od adresu *a*. Wprowadzenie jest kończone na podstawie licznika znaków albo po napotkaniu znaku cr (powrót karetki). Znak cr nie jest wprowadzany do pola. Wprowadzone znaki są kończone co najmniej jednym znakiem o kodzie 0. Liczba takich znaków nie przekracza 3.

WORD (*c* --)

Wykonanie operacji WORD zaczyna się od zdefiniowania dolnego bajtu danej słowowej *c* jako ogranicznika. Następnie pomijane są znaki ogranicznika występujące w strumieniu wejściowym, a znaki następne, poprzedzone jednobajtowym licznikiem znaków, są umieszczane w polu pamięci operacyjnej, rozpoczynającym się od pierwszego

wolnego bajtu słownika. Wprowadzanie kończy się po napotkaniu ogranicznika, znaku cr albo znaku o kodzie 0. Ostatnim znakiem umieszczonym w polu pamięci jest spacja.

QUERY (--)

Wykonanie operacji QUERY powoduje wprowadzenie z klawiatury do bufora terminala nie więcej niż 80 znaków. Wprowadzanie jest kończone na podstawie licznika znaków albo po napotkaniu znaku cr. Znak cr nie jest umieszczany w buforze terminala. Wprowadzone znaki są kończone co najmniej jednym znakiem o kodzie 0. Liczba takich znaków nie przekracza 3.

NUMBER (*a* -- *db*)

Wykonanie operacji NUMBER powoduje konwersję liczby zapisanej za pomocą ciągu znaków kodu ASCII na daną dwusłowową *db*. Przyjmuje się, że rozpatrywany ciąg znaków jest umieszczony w pamięci operacyjnej począwszy od adresu *a* i rozpoczyna się od bajtu określającego rozmiar ciągu poddawanego konwersji. Jeśli w rozpatrywanym ciągu znaków występują znaki . (kropka), to pozycja ostatniego z takich znaków zostanie zapamiętana w zmiennej indentyfikowanej przez operator DPL.

Przykład

Zdefiniowanie operatora określającego parzystość liczby pochodzącej z klawiatury terminala

```
: PARITY 0 BLK ! 0 WORD HERE NUMBER
         ." = " DROP 1 AND IF
            ." ODD"
         ELSE
            ." EVEN"
         THEN
         CR ;
```

• Wykonanie sekwencji 0 BLK ! powoduje nadanie zmiennej identyfikowanej przez operator BLK wartości 0, tj. utożsamienie strumienia wejściowego z ciągiem znaków pochodzących z terminala.

• Wykonanie operacji 0 WORD powoduje wprowadzenie z terminala jednego wiersza tekstu i umieszczenie go w buforze słowa.

• Wykonanie operacji NUMBER powoduje konwersję liczby wprowadzonej do bufora słowa.

• Jeśli np. zostanie wprowadzony z terminala napis PARITY 44, to na ekranie pojawi się tekst

PARITY 44 = EVEN □

Wyprowadzenie tekstów wieloznakowych umożliwiają operatory TYPE i COUNT, wyprowadzanie liczb operatory . (kropka), .R, D. i D.R, a redagowanie liczb zestaw operatorów

<# # #S #> HOLD SIGN

TYPE (*a n* --)

Wykonanie operacji TYPE powoduje wyprowadzenie *n* znaków z pola pamięci operacyjnej rozpoczynającego się od adresu *a*.

COUNT (*a* -- *b n*)

Wykonanie operacji COUNT powoduje zastąpienie — na stosie parametrów — adresu *a* ciągu znakowego rozpoczynającego się od licznika znaków, adresem $b = a + 1$, pierwszego znaku za licznikiem oraz samym licznikiem *n*. Operacja COUNT może być wykorzystana do przygotowania parametrów dla operacji TYPE.

<# (--)

Wykonanie operacji <# powoduje wykonanie czynności wstępnych przed konwersją danej dwusłowowej.

(*a* -- *b*)

Wykonanie operacji # powoduje umieszczenie — w buforze liczby — najmniej znaczącej cyfry danej dwusłowowej bez znaku *a* i zastąpienie jej — na stosie parametrów — daną dwusłowową *b* stanowiącą rezultat dzielenia danej *a* przez bieżącą podstawę liczb.

#S (*a* -- *b*)

Wykonanie operacji #S powoduje umieszczenie — w buforze liczby — wszystkich cyfr danej dwusłowowej bez znaku *a* i zastąpienie jej — na stosie parametrów — daną dwusłowową *b* o wartości 0.

#> (*v* -- *a n*)

Wykonanie operacji #> powoduje zakończenie redagowania liczby i udostępnienie — na stosie parametrów — adresu *a* i liczby znaków *n* ciągu znaków w kodzie ASCII, stanowiącego rezultat redagowania.

HOLD (*c* --)

Wykonanie operacji HOLD powoduje umieszczenie w buforze liczby znaku reprezentowanego w dolnym bajcie danej słowowej *c*.

SIGN (*n v* -- *v*)

Wykonanie operacji SIGN powoduje umieszczenie w buforze liczby znaku — (minus), ale tylko wtedy, gdy dana słowowa *n* ma wartość ujemną. Następnie dana słowowa *n* i dana dwusłowowa *v* zostają zastąpione na stosie parametrów daną *v*.

Przykład

Zdefiniowanie operatora PHONE wykorzystywanego do wyprowadzania numeru telefonu w postaci *dd-dd-dd*.

```
: PHONE
  <#  #  #  2D  HOLD
      #  #  2D  HOLD
  #S   #>   TYPE ;
```

● Napis 2D jest liczbą w kodzie szesnastkowym, reprezentującą znak – (minus) w kodzie ASCII.

● Kolejne operatory # i operator #S dotyczą cyfr liczby rozpatrywany od ostatniej do pierwszej. W szczególności, wykonanie sekwencji <# 2D HOLD # #S #> spowodowałoby wyprowadzenie liczby zakończonej minusem, a nie rozpoczynającej się od znaku minus.

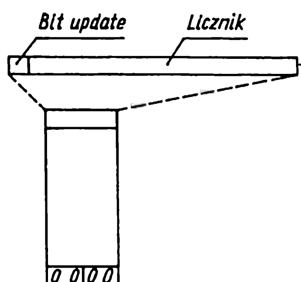
● Jeśli z klawiatury terminala zostanie np. wprowadzony napis 495.086 PHONE, to na ekranie pojawi się tekst 495.086 PHONE 49-50-86. □

14. Zarządzanie pamięcią zewnętrzną

W zestawie operatorów języka Forth występują takie, które dotyczą *pamięci zewnętrznej*. W tych implementacjach, w których nie przewidziano zewnętrznej *pamięci masowej*, są realizowane operacje na *pamięci symulowanej*. Dzięki temu zostają zachowane zasady współpracy zewnętrznej, a co najwyżej zmniejsza się jej rozmiar.

W większości implementacji przyjmuje się, że pamięć zewnętrzną stanowią dwie grupy bloków związanych ze stacjami dyskowymi identyfikowanymi przez operatory DR0 i DR1. W odwołaniach do bloków mogą być używane ich numery względne, liczone od 0 dla każdej grupy osobno, albo numery bezwzględne liczone od 0 względem zerowego bloku związanego ze stacją identyfikowaną przez DR0. Rozmiar każdego bloku może być określony za pomocą operatora B/BUF, a adres bezwzględny zerowego bloku dowolnej z grupy bloków — za pomocą operatora OFFSET.

Bloki sprowadzone do pamięci operacyjnej są przechowywane w buforach, po jednym bloku w buforze. Tak jak pokazano na rys. 14.1, każdy bufor jest zakończony parą znaków o kodzie 0 i rozpoczyna się daną słowową, która dzieli się na dwie części: bit *update* i numer danego bloku. Jeśli bit *update* — ustawiony za pomocą operacji UPDATE — ma wartość 1, to oznacza, że dany blok został zmodyfikowany w pamięci operacyjnej i wymaga zaktualizowania w pamięci dyskowej. Służy do tego operator FLUSH. Sprowadzenie bloku do pamięci operacyjnej umożliwia operator BLOCK, natomiast zarezerwowanie bufora dla bloku o podanym numerze, operator BUFFER. Należy mieć na uwadze to, iż parametrem operatora BLOCK jest numer względny bloku, natomiast parametrem operatora BUFFER jest numer bezwzględny. Ma to znaczenie w przypadku posługiwania się stacją dyskową identyfikowaną przez operator DR1.



14.1. Bufor dyskowy

OFFSET (-- *a*)
Wykonanie operacji OFFSET powoduje umieszczenie – na stosie parametrów – adresu *a* zmiennej określającej numer bezwzględny zerowego bloku bieżącej stacji dyskowej.

DR0 (--)
Wykonanie operacji DR0 czyni stację nr 0 stacją bieżącą i powoduje nadanie zmiennej identyfikowanej przez operator OFFSET wartości równej bezwzględnemu numerowi zerowego bloku dysku nr 0.

DR1 (--)
Wykonanie operacji DR1 czyni stację nr 1 stacją bieżącą i powoduje nadanie zmiennej identyfikowanej przez operator OFFSET wartości równej bezwzględnemu numerowi zerowego bloku dysku nr 1.

B/BUF (-- *n*)
Wykonanie operacji B/BUF powoduje umieszczenie – na stosie parametrów – danej słowowej określającej rozmiar bufora dyskowego, wyrażony w bajtach.

BLOCK (*n* -- *a*)
Wykonanie operacji BLOCK zaczyna się od sprawdzenia, czy w jednym z buforów dyskowych w pamięci operacyjnej znajduje się już blok o numerze względnym *n*. Jeśli go tam nie ma, to zostanie sprowadzony z pamięci dyskowej. Następnie dana słowowa *n*, znajdująca się na stosie parametrów, zostanie zastąpiona adresem *a* pierwszego bajtu danych tego bufora w pamięci operacyjnej, w którym znajduje się blok dyskowy o numerze *n*.

BUFFER (*n* -- *a*)
Wykonanie operacji BUFFER powoduje zarezerwowanie bufora w pamięci operacyjnej dla bloku dyskowego o numerze bezwzględnym

n. Następnie dana słowowa *n*, znajdująca się na stosie parametrów, zostanie zastąpiona adresem *a* pierwszego bajtu danych zarezerwowanego bufora.

UPDATE (--)
Wykonanie operacji UPDATE powoduje ustawienie bitu update w pierwszym bicie ostatnio użytego bufora dyskowego w pamięci operacyjnej.

Przykład

Skopiowanie zawartości bloku nr 20 pewnej stacji dyskowej do bloku nr 30 tej samej stacji

```
30 OFFSET @ + BUFFER UPDATE
20 BLOCK
SWAP B/BUF CMOVE
FLUSH
```



W tych przypadkach, gdy pamięć masowa jest rozpatrywana jako zestaw ekranów, znacznie wygodniej jest posługiwać się operatorem (LINE) oraz w ograniczonym zakresie operatorami B/SCR i SCR.

(LINE) (*l s* -- *a n*)
Wykonanie operacji (LINE) zapewnia sprowadzenie do pamięci operacyjnej tego bloku pamięci zewnętrznej, który zawiera wiersz *l* ekranu *s*.

```
VOCABULARY EDITOR IMMEDIATE
EDITOR DEFINITIONS
: TEXT HERE C/L 1+ BL FILL
       WORD HERE PAD C/L 1+ CMOVE ;
: LINE DUP FFF0 AND 17 ?ERROR
       SCR @ (LINE) DROP ;
: -MOVE LINE C/L CMOVE UPDATE ;
: H LINE PAD 1+ C/L DUP PAD C! CMOVE ;
: E LINE C/L BL FILL UPDATE ;
: S DUP 1 - 0E
    DO I LINE I 1+ -MOVE -1 +LOOP E ;
: D DUP H OF DUP ROT
    DO I 1+ LINE I -MOVE LOOP E ;
: R PAD 1+ SWAP -MOVE ;
: P 1 TEXT R ;
: I DUP S R ;
: CLEAR SCR ! 10 0
        DO FORTH I EDITOR E LOOP ;
```

14.2. Definicje operatorów podsłownika EDITOR

Po wykonaniu tej czynności, na stosie parametrów pozostają: adres *a* pierwszego bajtu wiersza w buforze dyskowym w pamięci operacyjnej i rozmiar tego wiersza w znakach *n*.

B/SCR (-- *n*)
Wykonanie operacji B/SCR powoduje umieszczenie — na stosie parametrów — danej słowowej *n* określającej z ilu bloków dyskowych składa się jeden ekran.

SCR (-- *a*)
Wykonanie operacji SCR powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej określającej numer bloku, którym posłużono się w ostatnio wykonanej operacji LIST.

Biorąc pod uwagę przytoczone opisy, na rys. 14.2 przedstawiono zestaw definicji niektórych operatorów edycyjnych podsłownika EDITOR.

15. Grafika i animacja

Operacje umożliwiające oprogramowanie *grafiki* i *animacji* nie wchodzą w zakres języka Forth. Jest to w głównej mierze spowodowane różnorodnością sprzętu komputerowego, zarówno procesorów jak i urządzeń zewnętrznych. Tym niemniej, w większości implementacji dostępne są operacje podstawowe umożliwiające wykreślanie linii i punktów. Znacznie rzadziej są natomiast dostępne operacje umożliwiające programowanie animacji.

W celu zilustrowania sposobu implementowania podstawowych operacji graficznych na rys. 15.1 przedstawiono definicje operatorów CLEAR, PLOT i DRAW. Definicje te zostały umieszczone w podsłowniku GRAPHICS i dotyczą mikrokomputera Amstrad 6128 działającego w trybie z wysoką rozdzielczością 640×200 pixeli. Dla skrócenia opisu tylko jeden z operatorów zdefiniowano w kodzie maszynowym. W definicji tej posłużono się mnemonikami asemblera dla mikroprocesora Intel 8080. Takie postępowanie jest dopuszczalne, ponieważ użyty w Amstradzie mikroprocesor Z80 jest zgodny z tym mikroprocesorem.

CLEAR (--)

Wykonanie operacji CLEAR powoduje wygaszenie ekranu i ustalenie pozycji bieżącej w lewym dolnym rogu ekranu. Pozycja ta ma współrzędne (0,0).

PLOT (*x y* --)

Wykonanie operacji PLOT powoduje uczynienie pozycją bieżącą pozycji o współrzędnych (*x*,*y*), a następnie umieszczenie na tej pozycji punktu świecącego (*piksela*).

DRAW (*x y* --)

Wykonanie operacji DRAW powoduje wykreślenie odcinka, składającego się z punktów świecących, od punktu określonego przez pozycję bieżącą do punktu o współrzędnych (*x*,*y*).

Definicja operacji CLEAR wykorzystuje fakt, iż poszczególne punkty wyświetlane na ekranie stanowią odwzorowanie bitów *bufora ekranu* w pamięci operacyjnej. Bufor ekranu zajmuje zazwyczaj obszar od C000 (−16384) do FFFF (−1) i jest podzielony na 8 stref o rozmiarze 2048 bajtów każda. Z tych 2048 bajtów wykorzystuje się tylko pierwsze 2000 bajtów. Jeśli przyjąć, że ekran jest podzielony w pionie na 25 pasków po 8 wierszy każdy, to pierwsze wiersze każdego paska następują po sobie w pierwszej strefie bufora ekranu, drugie wiersze w drugiej strefie, itd. Wynika stąd, że wyzerowanie kolejnych wierszy ekranu, od góry do dołu, wymaga cyklicznego przemieszczania się między kolejnymi strefami i zerowania w każdej z nich po 80 kolejnych bajtów. Należy także zwrócić uwagę, że definicja operatora CLEAR odwołuje się do pomocniczych operatorów X i Y identyfikujących zmienne określające współrzędne pozycji bieżącej.

Definicja operatora PLOT odwołuje się do pomocniczego operatora P, a także do operatorów X i Y. Wykonanie operacji P (*x y* −− *a m*) powoduje zastąpienie − na stosie parametrów − danych *x* i *y* określających współrzędne pozycji bieżącej adresem *a* bajtu i maską *m* określającą pozycję w bajcie, dla punktu o wypółrzędnych (*x*,*y*). Jak można się przekonać

$$a = C000+((199-y)\ /\ 8)*80+$$
$$+((199-y)\text{mod }8)*2048+x\ /\ 8$$
$$m = 128>>(x \text{ mod } 8)$$

gdzie / oznacza dzielenie całkowite, a > > przesunięcie w prawo. W celu ułatwienia analizy przytoczonych wzorów można podać, że

C000
: jest adresem bufora ekranu;

$199-y$
: jest numerem wiersza liczonym od góry ekranu;

$(199-y)\ /\ 8$
: jest numerem paska, liczonym od góry ekranu, w którym znajduje się punkt (*x*,*y*);

$((199-y)\ /\ 8) * 80$
: jest adresem bajtu, liczonym od początku bufora ekranu, od którego zaczyna się pierwszy wiersz paska zawierającego punkt (*x*,*y*);

$(199-y) \text{ mod } 8$
: jest numerem wiersza w obrębie paska zawierającego punkt (*x*,*y*);

$((199-y) \text{ mod } 8)*2048$
: jest różnicą dwóch adresów: adresu wiersza, w którym znajduje się punkt (*x*,*y*) i adresu paska, w którym znajduje się ten punkt;

$x\ /\ 8$
: jest numerem bajtu w obrębie wiersza zawierającego punkt (*x*,*y*);

$x \text{ mod } 8$
: jest numerem bitu w obrębie bajtu zawierającego punkt (*x*,*y*).

```
VOCABULARY  GRAPHICS  IMMEDIATE
GRAPHICS  DEFINITIONS
DECIMAL

: /8  ( v — v 8 )  8  /  ;
: *2048  ( v — v*2048 )  2048  *  ;
: *8  ( v — v*8 )  8  *  ;
: *80  ( v — v*80 )  80  *  ;
     CODE BIT  ( n — b )
          D POP                    (         POP  D     )
          E INR                    (         INR  E     )
          A XRA                    (         XRA  A     )
          128 ORI                  (         ORI  128   )
          HERE  E DCR              ( loop :  DCR  E     )
          HERE  1+  SWAP  0 JZ     (         JZ   skip  )
          RRC                      (         RRC        )
          JMP                      (         JMP  loop  )
          HERE  SWAP  !  E A MOV   ( skip :  MOV  E,A   )
          D PUSH                   (         PUSH D     )
          C;
0  VARIABLE  X
0  VARIABLE  Y

 : CLEAR  -16384  0  8  DO
             DUP  DUP  2000  +  DO
                0  I  C!
             LOOP
             2048  +
          LOOP
          DROP  ;
          0  X  !  0  Y  !  ;
 : P  ( x y — a m )
      MINUS  199  +  DUP  /8  *80  >R
      7  AND  *2048  R>  +  OVER  /8  +
      -16384  +  SWAP  7  AND  BIT  ;
 : PLOT  ( x y — )
         OVER  OVER  Y  !  X  !  P
         OVER  C@  XOR  SWAP  C!  ;
 : T  ( c2 c1 — s a )
      -  DUP  DUP  0<  IF
         DROP  -1
      ELSE
         0=  0=
      THEN
      SWAP  ABS  ;
```

```
  0  CONSTANT  F
  2  CONSTANT  DX
  4  CONSTANT  DY
  6  CONSTANT  S1
  8  CONSTANT  S2
 10  CONSTANT  E
  0  VARIABLE  VAR  10  ALLOT  VAR  CONSTANT  H
  : W@  H  +  @  ;
  : W!  H  +  !  ;
  : DRAW  ( x2  y2  -- )
       DUP  >R  OVER  >R
       Y  @  DUP  >R  T
          DY  W!  S2  W!
       X  @  DUP  >R  T
          DX  W!  S1  W!
       R>  R>  R>  X  !  R>  Y  !
       DY  W@  DX  W@  OVER  OVER  +  0=  IF
          DROP  DROP  PLOT
     ELSE
        OVER  OVER  >  DUP  F  W!  IF
           DY  W!  DX  W!
        ELSE
           DROP  DROP
        THEN
        DY  W@  DUP  +  DX  W@  -  E  W!
        DX  W@  0  DO
           OVER  OVER  PLOT
           BEGIN
              E  W@  0<  0=  WHILE
              F  IF
                 SWAP  S1  W@  +  SWAP
              ELSE
                 S2  W@  +
              THEN
              E  W@  DX  W@  DUP  +  -  E  W!
           REPEAT
           F  W@  IF
              S2  W@  +
           ELSE
              SWAP  S1  W@  +  SWAP
           THEN
              E  W@  DY  W@  DUP  +  +  E  W!
       LOOP
     THEN  ;
FORTH  DEFINITIONS
GRAPHICS
```

15.1. Definicje operatorów graficznych

Wykreślenie odcinka linii prostej między punktami o współrzędnych (x_1,y_1) i (x_2,y_2)

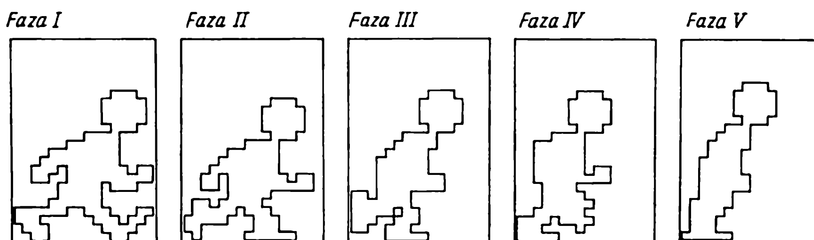
```
y = y1
x = x1
dy = abs(y2 - y1)
s2 = sign(y2 - y1)
dx = abs(x2 - x1)
s1 = sign(x2 - x1)
if dy + dx = 0 then
   plot(x,y)
else
   f = dy > dx
   if f then
      t = dx
      dx = dy
      dy = t
   end if
   e = 2 * dy - dx
   for i = 1 to dx
      plot(x,y)
      while(e ≥ 0)
         if f then
            x = x + s1
         else
            y = y + s2
         end if
         e = e - 2 * dx
      end while
      if f then
         y = y + s2
      else
         x = x + s1
      end if
      e = e + 2 * dy
   end for
end if
```

15.2. Algorytm Bresenhama dla odcinka

W definicji operatora DRAW występuje odwołanie do operatora PLOT oraz odwołania do szeregu operatorów pomocniczych. Definicja ta realizuje uogólniony algorytm Bresenhama do kreślenia odcinków linii prostych metodą przyrostową. Algorytm ten przedstawiono na rys. 15.2.

Programowanie animacji, tj. przygotowywanie definicji umożliwiających uzyskiwanie efektów ruchowych, jest znacznie bardziej złożone niż programowanie grafiki. W ogólnym przypadku polega ono na wyznaczaniu różnicy symetrycznej układu bitów ekranu z układem bitów obiektu animowanego. Różnica ta jest wyznaczana dwukrotnie dla każdej nowej pozycji obiektu animowanego, dzięki czemu obiekt ten na przemian pojawia się na ekranie i znika. Jeżeli po zniknięciu, obraz obiektu zostanie wygenerowany w sąsiednim miejscu, to uzyska się efekt ruchu. Posługiwanie się różnicą symetryczną jest podyktowane wymaganiem nie naruszenia tła.

Na rysunku 15.3 przedstawiono pewien obiekt w pięciu fazach ruchu, a na rys. 15.4 zbiór definicji umożliwiających animację tego obiektu. W szczególności, po zdefiniowaniu przez kompilator DEF operatorów E1, E2, E3, E4 i E5 każde wykonanie operacji RUN powoduje uzyskanie efektów ruchowych.



15.3. Obiekt animowany w 5 fazach ruchu

Znaczenie poszczególnych operatorów jest następujące:

DEF (48-danych-słowowych — —)

Wykonanie operacji DEF powoduje utworzenie definicji operatora o nazwie identycznej z nazwą najbliższego słowa występującego w strumieniu wejściowym, a następnie umieszczenie w polu parametrów tego operatora 48 danych bajtowych określających kształt obiektu identyfikowanego przez dany operator. Późniejsze wykonanie tak zdefiniowanej operacji w kontekście

a nazwa

spowoduje umieszczenie obiektu w środkowej części ekranu. Dokona się to w taki sposób, że lewy górny piksel obiektu znajdzie sę w tym bajcie bufora ekranu, który ma adres *a*.

```
GRAPHICS DEFINITIONS
VOCABULARY ANIMATION IMMEDIATE
ANIMATION DEFINITIONS
: DEF <BUILDS 0 48 DO C, LOOP DOES>
        3 0 DO
           OVER OVER
           8 0 DO
              OVER OVER C@ SWAP C!
              OVER 1+ OVER 1+ C@ SWAP C!
              2+ SWAP 2048 + SWAP
           LOOP
           DROP DROP
           16 + SWAP 80 + SWAP
        LOOP
       .DROP DROP ;
HEX
( FAZA I )
00 00 00 00 00 00 00 00 00 00 00 00 00 1C 00 3E
00 3E 00 3E 00 1C 00 F0 03 F0 0F F0 1D F0 3B FB
33 FF 07 C8 07 C0 0F C0 FC E3 E0 76 70 3C 30 18 DEF E1
( FAZA II )
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 38
00 7C 00 7C 00 7C 00 38 01 E0 07 E0 0F E0 1F E0
37 F6 37 FE 07 C0 6F 80 7D C0 F8 E0 C0 E0 41 F8 DEF E2
( FAZA III )
00 00 00 00 00 00 00 00 00 00 00 00 00 70 00 F8
00 F8 00 F8 00 70 03 C0 07 C0 0F 80 1F 80 1F 80
1F E0 1F E0 DE 00 FE 00 FB 00 E7 00 CE 00 0F 80 DEF E3
( FAZA IV )
00 00 00 00 00 00 00 00 00 00 00 00 00 E0 01 F0
01 F0 01 F0 00 E0 07 80 1F 80 1F 80 1F 80 1F B0
1F F0 0F 00 0F 80 0D C0 1F 80 7B C0 70 80 7C 00 DEF E4
( FAZA V )
00 00 00 00 00 00 00 00 00 00 01 C0 03 E0 03 E0
03 E0 01 C0 07 00 0F 00 1F 00 1E 00 3E 00 3E 00
3F 00 3F 00 3C 00 7C 00 78 00 70 00 70 00 FC 00 DEF E5
DECIMAL
: ADR [ -16384 11 80 * + ] LITERAL + ;
: N 2+ DUP DUP ;
: RUN CLEAR 0 ADR DUP DUP
       0 40 DO
          E1 E1 N
          E2 E2 N
          E3 E3 N
          E4 E4 N
          E5 E5 N
       LOOP ;
```

15.4. Definicje operatorów animacyjnych

ADR (*k* -- *a*)
Wykonanie operacji ADR powoduje przekształcenie numeru kolumny *k* na adres bajtu, w którym znajduje się lewy górny piksel znaku umieszczonego na pozycji (11,*k*). Pozycje znaków są liczone od lewego górnego narożnika ekranu. Wiersz liczy 80, a kolumna 25 pozycji. Znak znajdujący się w narożniku ekranu zajmuje pozycję (0,0).

Część II

Opisy operacji

Przytoczone w tej części książki szczegółowe opisy operacji przedstawiono w porządku alfabetycznym wyznaczonym przez alfabet ASCII. W zamieszczonych definicjach posłużono się liczbami w zapisie szesnastkowym. W celu ułatwienia wyszukiwania opisów operacji bezpośrednio po skorowidzu umieszczono indeks operatorów.

! (*s a* --)

Wykonanie operacji ! (*store*) powoduje umieszczenie danej słowowej *s* pod adresem *a*. ! jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

!CSP (--)

Wykonanie operacji !CSP (*store-C-S-P*) powoduje przypisanie zmiennej identyfikowanej przez operator CSP adresu szczytu stosu parametrów. !CSP jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: !CSP SP@ CSP ! ;
```

(*dv* -- *dq*)

Wykonanie operacji # (*sharp*) powoduje wyodrębnienie najmniej znaczącej cyfry danej dwusłowowej *dv*. Znak ASCII reprezentujący tę cyfrę jest umieszczany w kolejnym wolnym bajcie bufora liczby. Konwersja odbywa się z uwzględnieniem bieżącej podstawy liczb, a daną *dq* stanowi iloraz danej *dv* i bieżącej podstawy. Operator # występuje z reguły w ciągu między operatorami <# i #>. # jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: # BASE @ M/MOD ROT 9 OVER <
    IF 7 + THEN 30 +HOLD ;
```

#> (*dv* — — *a c*)

Wykonanie operacji #> (*sharp-greater-than*) powoduje zakończenie konwersji danej dwusłowowej na ciąg znaków ASCII. Dana *dv* zostaje odrzucona i zastąpiona na stosie parametrów adresem *a* — najbardziej znaczącego znaku powstałego z konwersji oraz licznikiem znaków *c*. Operator #> występuje z reguły jako ostatni z operatorów ciągu <# ... #>. #> jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: #>  DROP  DROP  HLD  @  PAD  OVER  -  ;
```

#S (*dv* — — 0 0)

Wykonanie operacji #S powoduje dokończenie konwersji danej dwusłowowej bez znaku *dv* na ciąg ASCII. Kolejno otrzymywane znaki kodu ASCII są umieszczane w wolnych bajtach bufora liczby. Operator #S występuje z reguły między operatorami <# i #>. #S jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: #S  BEGIN
          #  OVER  OVER  OR  0=
      UNTIL  ;
```

' (— — *a*) — stan wykonywania
(— —) — stan kompilacji

Wykonanie operacji ' (*tick*) zaczyna się od rozpatrzenia stanu interpretera. Jeśli interpreter znajduje się w stanie wykonywania, to na stosie parametrów zostanie umieszczony adres pola parametrów operatora o nazwie identycznej z nazwą najbliższego słowa występującego w strumieniu wejściowym. Jeśli interpreter znajduje się w stanie definiowania, to w słowniku zostanie umieszczone wskazanie operatora LIT, a za nim wspomniany adres pola parametrów operatora. W obu przypadkach słowo, które określało nazwę operatora zostanie pominięte. ' jest operatorem czynnym zdefiniowanym, za pomocą operatora : (dwukropek)

```
: '  -FIND  0=  0  ERROR  DROP
   [COMPILE]  LITERAL  ;  IMMEDIATE
```

((— —)

Wykonanie operacji ((*left-paren*) powoduje pominięcie wszystkich następnych znaków interpretowanego tekstu wejściowego, aż do najbliższego znaku) włącznie, tj. potraktowanie tego tekstu jako komentarza. (jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ( 29 WORD ; IMMEDIATE
```

(.") (— —) por. opis operacji ."

Wykonanie operacji (.") (*paren-dot-quote*) powoduje wyprowadzenie tekstu następującego po wskazaniu operatora (."). Tekst ten jest poprzedzony bajtem licznika określającym liczbę znaków tekstu. (.") jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: (.") R COUNT DUP 1+ R> + >R TYPE ;

(+LOOP) (v — —) por. opis operacji +LOOP

Wykonanie operacji (+LOOP) (*paren-plus-loop*) powoduje dodanie do zmiennej sterującej cyklu danej v i podjęcie decyzji o ewentualnym dalszym wykonywaniu cyklu. Jeśli dana v ma wartość dodatnią, to cykl zostanie zakończony, gdy nowa wartość zmiennej sterującej zrówna się z ograniczeniem albo je przekroczy. Jeśli dana v ma wartość ujemną, to cykl zostanie zakończony, gdy nowa wartość zmiennej sterującej równa się z ograniczeniem albo okaże się od niego mniejsza. Zabrania się, aby dana v miała wartość 0. (+LOOP) jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

(;CODE) (— —) por. opis operacji ;CODE

Wykonanie operacji (;CODE) (*paren-semicolon-code*) powoduje zmianę pola kodu kompilowanego operatora w taki sposób, że zawierać będzie ono adres kodu maszynowego następującego po wskazaniu operatora (;CODE). (;CODE) jest operatorem biernym, zdefiniowanym za pomocą operatora : (dwukropek)

: (;CODE) R> LATEST PFA CFA ! ;

(ABORT) (cały-stos — —)

Wykonanie operacji (ABORT) (*paren-abort*) powoduje wykonanie takich czynności jak podczas wykonywania operacji ABORT. (ABORT) jest operatorem biernym, zdefiniowanym za pomocą operatora : (dwukropek)

: (ABORT) ABORT ;

(DO) (l i — —) por. opis operacji DO

Wykonanie operacji (DO) (*paren-do*) powoduje ustalenie parametrów cyklu DO ... LOOP albo DO ... +LOOP. W typowych implementacjach parametry te zostaną umieszczone na stosie powrotów; najpierw parametr l (ograniczenie), a następnie parametr i (zmienna sterująca). (DO) jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

(FIND) (*a b* -- *p n t*)
(*a b* -- *f*)

Wykonanie operacji (FIND) (*paren-find*) powoduje przeszukanie słownika w celu znalezienia definicji operatora. Przyjmuje się, że pole nazwy pierwszej rozpatrywanej definicji operatora rozpoczyna się od adresu *b*, a pole określające nazwę poszukiwanego słowa rozpoczyna się od adresu *a*. Pierwszy bajt tego pola zawiera licznik znaków nazwy. Jeśli definicja operatora zostanie znaleziona, to na stosie parametrów pozostaną: adres *p* pola parametrów odszukanej definicji, liczba znaków *n* nazwy operatora oraz dana *t* o wartości 1. W przeciwnym razie, na stosie parametrów pozostanie jedynie dana *f* o wartości 0. (FIND) jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

(LINE) (*n s* -- *a l*)

Wykonanie operacji (LINE) (*paren-line*) powoduje przekształcenie współrzędnych wiersza tekstu przechowywanego w pamięci zewnętrznej na jego współrzędne w pamięci operacyjnej. Dana *n* określa numer wiersza w obrębie ekranu, a dana *s* – numer ekranu. Po sprowadzeniu do pamięci operacyjnej bloku zawierającego tak określony wiersz na stosie parametrów pozostaną adres *a* pierwszego bajtu wiersza w buforze w pamięci operacyjnej oraz rozmiar wiersza w znakach *l*. (LINE) jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: (LINE) >R C/L B/BUF */MOD
         R>  B/SCR * + BLOCK + C/L ;
```

(LOOP) (--) por. opis operacji LOOP

Wykonanie operacji (LOOP) (*paren-loop*) powoduje zwiększenie o 1 zmiennej sterującej cyklu i podjęcie decyzji o ewentualnym dalszym wykonywaniu cyklu. Cykl zostanie uznany za zakończony, gdy nowa wartość zmiennej sterującej zrówna się z ograniczeniem albo je przekroczy. (LOOP) jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

(NUMBER) (*da a* -- *db b*)

Wykonanie operacji (NUMBER) (*paren-number*) powoduje wykonanie konwersji liczby zapisanej w kodzie ASCII na daną dwójkową podwojonej dokładności. Konwersja uwzględnia bieżącą podstawę liczb. Dana ta reprezentuje rezultat dotychczasowej konwersji, *a* jest adresem bajtu poprzedzającego pierwszą cyfrę liczby poddawanej konwersji, *db* reprezentuje rezultat uaktualnionej konwersji, *b* zaś jest

adresem pierwszego znaku liczby, który nie jest cyfrą przy danej podstawie. (NUMBER) jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: (NUMBER) BEGIN
              1+ DUP >R C@ BASE @ DIGIT
           WHILE
              SWAP BASE @ U* DROP ROT BASE @
              U* D+ DPL @ 1+
              IF 1 DPL +! THEN
              R>
           REPEAT R> ;
```

* (*a b* -- *p*)

Wykonanie operacji * (times) powoduje zastąpienie — na stosie parametrów — danych *a* i *b* ich iloczynem *p*. * jest operatorem biernym zdefiniowanym za pomocą kompilatora : (dwukropek)

: * M* DROP ;

*/ (*a b c* -- *q*)

Wykonanie operacji */ (*times-divide*) powoduje zastąpienie — na stosie parametrów — danych *a*, *b* i *c* daną o wartości *a* * *b* / *c*. Rezultat obliczeń jest dokładniejszy niż w przypadku operacji *a b* * *c* / ponieważ rezultatem iloczynu jest dana podwojonej dokładności. */ jest operatorem biernym zdefiniowanym za pomocą kompilatora : (dwukropek)

: */ */MOD SWAP DROP ;

*/MOD (*a b c* -- *r q*)

Wykonanie operacji */MOD (*times-divide-mod*) powoduje zastąpienie — na stosie parametrów — danych *a*, *b* i *c* parą danych *r* i *q* tak dobranych, że *q* jest daną o wartości *a* * *b* / *c*, zaś *r* jest resztą z dzielenia *a* * *b* przez *c*. */MOD jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: */MOD >R M* R> M/ ;

\+ (*a b* -- *s*)

Wykonanie operacji + (plus) powoduje zastąpienie — na stosie parametrów — danych *a* i *b* ich sumą $s = a+b$. + jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

+! (*v a* --)

Wykonanie operacji +! (*plus-store*) powoduje dodanie do danej słowowej znajdującej się pod adresem *a* danej o wartości *v*. +! jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

+– (*a b* –– *c*)
Wykonanie operacji +– (*plus-minus*) powoduje zastąpienie – na stosie parametrów – danych *a* i *b* daną *c*. Jeśli *b* jest ujemne, to *c* = –*a*. W przeciwnym razie *c* = *a*. +– jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: +- 0< IF MINUS THEN ;
```

+BUF (*a* –– *b f*)
Wykonanie operacji +BUF (*plus-buf*) powoduje zastąpienie – na stosie parametrów – adresu *a*, bufora dyskowego zawartego w pamięci operacyjnej, adresem *b* następnego bufora oraz znacznikiem *f*. Jeśli adres tego następnego bufora stanowi wartość zmiennej PREV to znacznik *f* jest daną o wartości 0. W przeciwnym razie znacznik jest daną o wartości różnej od 0. +BUF jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: +BUF B/BUF 4 + + DUP LIMIT =
       IF DROP FIRST THEN
       DUP PREV @ - ;
```

+LOOP (*a n*3 ––) por. opis operacji DO
Wykonanie operacji +LOOP (*plus-loop*) zaczyna się od upewnienia, że dana *n*3 ma wartość 3. Jeśli tak nie jest, to operacja +LOOP jest użyta w niepoprawnym kontekście. Spowoduje to wykonanie czynności przedstawionych w opisie operatora ?PAIRS. W przeciwnym razie w słowniku zostanie umieszczone wskazanie operatora (+LOOP), a za nim adres względny tego operatora, który został umieszczony w słowniku pod adresem *a*. +LOOP jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: +LOOP 3 ?PAIRS COMPILE (+LOOP)
      BACK; IMMEDIATE
```

+ORIGIN (*n* –– *a*)
Wykonanie operacji +ORIGIN (*plus-origin*) powoduje zastąpienie – na stosie parametrów – adresu względnego *n* w obszarze użytkownika (*user area*) adresem rzeczywistym *a*. +ORIGIN jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: +ORIGIN origin + ;
```

(napis *origin* reprezentuje adres rzeczywisty początku obszaru użytkownika).

, (*v* — —)

Wykonanie operacji , (*comma*) powoduje przeniesienie danej słowowej ze stosu parametrów do słownika. , jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: , HERE ! 2 ALLOT ;

— (*a b* — — *s*)

Wykonanie operacji — (*minus*) powoduje zastąpienie — na stosie parametrów — danych *a* i *b* ich różnicą $s = a - b$. — jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: — MINUS + ;

— — > (— —)

Wykonanie operacji — — > (*next-screen*) powoduje kontynuowanie interpretacji strumienia wejściowego począwszy od następnego ekranu. Operator — — > może być użyty tylko podczas interpretowania ekranu i nie może być użyty w obrębie definicji nowego operatora. — — > jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: — — > ?LOADING 0 IN ! B/SCR BLK @ OVER
MOD — BLK +! ; IMMEDIATE

—DUP (*a* — — *a*) — dla $a = 0$
(*a* — — *a a*) — dla $a \neq 0$

Wykonanie operacji — DUP (*dash-dupe*) powoduje warunkowe powielenie danej znajdującej się na szczycie stosu parametrów. Powielenie to następuje jedynie wtedy, kiedy *a* ma wartość różną od zera. -DUP jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: —DUP DUP IF DUP THEN ;

—FIND (— — *a n t*)
(— — *f*)

Wykonanie operacji —FIND (*dash-find*) powoduje wprowadzenie ze strumienia wejściowego jednego słowa i przeszukanie słownika w celu znalezienia w nim definicji operatora o nazwie identycznej z nazwą tego słowa. Przeszukiwanie dotyczy części słownika określonej przez operator CONTEXT, a następnie części słownika określonej przez operator CURRENT. Jeśli definicja operatora zostanie znaleziona, to na stosie parametrów pozostaną: adres pola parametrów *a* odszukanej definicji, liczba znaków nazwy *n* wzięta z definicji oraz dana *t* o war-

tości 1. W przeciwnym razie na stosie parametrów pozostanie dana *f* o wartości 0. —FIND jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: —FIND  BL  WORD  HERE
         CONTEXT  @  @
         (FIND)  DUP  0=
         IF
           DROP  HERE  LATEST  (FIND)
         THEN  ;
```

—TRAILING (*a c* — — *a n*)

Wykonanie operacji —TRAILING (*dash-trailing*) powoduje zastąpienie — na stosie parametrów — pary danych *a* i *c*, określających odpowiednio adres i liczbę znaków ciągu znaków ASCII, parą danych *a* i *n*, z tak dobranym największym *n* nie większym niż *c*, aby ciąg znaków zaczynający się od adresu *a* i liczący *n* znaków nie zawierał spacji kończących. —TRAILING jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: —TRAILING  DUP  0
             DO
                OVER  OVER  +  1  —  C@  BL  —
                IF  LEAVE
                ELSE  1  —
                THEN
             LOOP  ;
```

(*v* — —)

Wykonanie operacji . (*dot*) powoduje wyprowadzenie wartości danej znajdującej się na szczycie stosu parametrów. Wartość ta jest przedstawiana jako wyrównana lewostronnie liczba całkowita, po której następuje znak spacji. Jeśli dana ma wartość ujemną, to pierwsza cyfra liczby jest poprzedzona znakiem — (minus). . jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: . S— —>D D. ;
```

." (— —)

Wykonanie operacji ." (*dot-quote*) może nastąpić zarówno w stanie definiowania jak i w stanie wykonywania. W pierwszym przypadku nastąpi umieszczenie w słowniku wskazania operatora (."), a za nim — — poprzedzonego licznikiem — ciągu znaków zawartego między operatorem ." a najbliższym znakiem " (cudzysłów) wyłącznie. W drugim przypadku nastąpi wyprowadzenie ciągu znaków między ."

i znakiem " (cudzysłów). ." jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ." 22 STATE @
   IF
     COMPILE (.") WORD HERE C@ 1+ ALLOT
   ELSE
     WORD HERE COUNT TYPE
   THEN ; IMMEDIATE
```

.LINE (*l s* --)

Wykonanie operacji .LINE (*dot-line*) powoduje wyprowadzenie wiersza o numerze *l*, zawartego w ekranie dyskowym o numerze *s*. .LINE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: .LINE (LINE) -TRAILING TYPE ;
```

.R (*v w* --)

Wykonanie operacji .R (*dot-R*) powoduje wyprowadzenie wartości danej *v* jako liczby wyrównanej prawostronnie w polu o szerokości *w*. .R jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: .R >R S-->D R> D.R ;
```

/ (*a b* -- *c*)

Wykonanie operacji / (*divide*) powoduje zastąpienie — na stosie parametrów — danych *a* i *b* ich ilorazem $c = a/b$. / jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: / /MOD SWAP DROP ;
```

/MOD (*a b* -- *r q*)

Wykonanie operacji /MOD (*divide-mod*) powoduje zastąpienie — na stosie parametrów — danych *a* i *b* parą danych *r* i *q* tak dobranych, że $q = a - b$, zaś *r* jest resztą z dzielenia *a* przez *b*. /MOD jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
:  /MOD >R S--> D R> M/ ;
```

0 (-- 0)

Wykonanie operacji 0 (*zero*) powoduje umieszczenie — na stosie parametrów — danej o wartości 0. 0 jest operatorem biernym, zdefiniowanym za pomocą kompilatora CONSTANT.

0< (*v* -- *t*)
(*v* -- *f*)
Wykonanie operacji 0< (*zero-less-then*) powoduje rozpatrzenie relacji $v < 0$. Jeśli relacja ta jest prawdziwa, to dana v zostanie zastąpiona daną t o wartości 1. W przeciwnym razie zostanie zastąpiona daną f o wartości 0. 0< jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

0= (*v* -- *t*)
(*v* -- *f*)
Wykonanie operacji 0= (*zero-equal*) powoduje rozpatrzenie relacji $v = 0$. Jeśli relacja ta jest prawdziwa, to dana v zostanie zastąpiona daną t o wartości 1. W przeciwnym razie dana v zostanie zastąpiona daną f o wartości 0. 0= jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

0BRANCH (*v* --)
Wykonanie operacji 0BRANCH (*zero-branch*) powoduje rozpatrzenie relacji $v = 0$. Jeśli relacja ta jest prawdziwa, to do licznika interpretera IP zostanie dodana wartość danej następującej bezpośrednio po wskazaniu operatora 0BRANCH. W przeciwnym razie licznik interpretera zostanie zwiększony o 2. Zrealizowanie takiej operacji jest równoważne wykonaniu warunkowego skoku względnego. 0BRANCH jest operatorem biernym zdefiniowanym w kodzie maszynowym.

1 (-- 1)
Wykonanie operacji 1 (*one*) powoduje umieszczenie – na stosie parametrów – danej o wartości 1. 1 jest operatorem biernym, zdefiniowanym za pomocą kompilatora CONSTANT.

1+ (*v* -- *n*)
Wykonanie operacji 1+ (*one-plus*) powoduje zastąpienie – na stosie parametrów – danej v daną n o wartości $v+1$. 1+ jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: 1+ 1 + ;
```

2 (-- 2)
Wykonanie operacji 2 (*two*) powoduje umieszczenie – na stosie parametrów – danej o wartości 2. 2 jest operatorem biernym, zdefiniowanym za pomocą kompilatora CONSTANT.

2+ (*v* -- *n*)
Wykonanie operacji 2+ (*two-plus*) powoduje zastąpienie – na stosie parametrów – danej v daną n o wartości $v+2$. 2+ jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: 2+ 2 + ;
```

3 (-- 3)

Wykonanie operacji 3 (*three*) powoduje umieszczenie — na stosie parametrów — danej o wartości 3. 3 jest operatorem biernym, zdefiniowanym za pomocą kompilatora CONSTANT.

: (--)

Wykonanie operacji : (*colon*) zaczyna się od upewnienia, iż interpreter znajduje się w stanie wykonywania. Jeśli tak nie jest, to operator : jest użyty w niepoprawnym kontekście. Spowoduje to wyprowadzenie komunikatu

EXECUTION ONLY

i wykonanie czynności przedstawionych łącznie w opisach operatorów ?ERROR i WARNING. W przeciwnym razie nastąpi utworzenie nagłówka definicji nowego operatora, o nazwie wyrażonej przez słowo występujące po operatorze : (dwukropek). Nagłówek będzie składać się z pola nazwy, pola łącznika i pola kodu i zostanie umieszczony w podsłowniku określonym przez zmienną identyfikowaną przez operator CURRENT. Tuż przed utworzeniem nagłówka nastąpi wykonanie operacji !CSP, tj. zapamiętanie bieżącej pozycji stosu parametrów. Umożliwi to przyszłą weryfikację, czy podczas definiowania operatora rozmiar stosu parametrów nie uległ zmianie. : jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: : ?EXEC !CSP CURRENT @ CONTEXT !
CREATE] ;CODE *kod-maszynowy* IMMEDIATE

(*kod-maszynowy* reprezentuje program w kodzie maszynowym, którego adres zostanie umieszczony w polu CFA każdego operatora skompilowanego za pomocą operatora : (dwukropek)).

; (--) por. opis operacji :

Wykonanie operacji ; (*semicolon*) zaczyna się od upewnienia, iż podczas kompilowania definicji nie nastąpiła zmiana rozmiaru stosu parametrów. Jeśli tak nie jest, to nastąpi wyprowadzenie komunikatu

DEFINITION NOT FINISHED

a następnie wykonanie czynności przedstawionych łącznie w opisach operatorów ?ERROR i WARNING. W przeciwnym razie nastąpi umieszczenie w słowniku wskazania operatora ;S, zanegowanie bitu *smudge* właśnie definiowanego operatora i ustawienie interpretera w stan wykonywania. ; jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: ; ?CSP COMPILE ;S SMUDGE
[COMPILE] [; IMMEDIATE

;CODE (--) por. opis operacji :

Wykonanie operacji ;CODE (*semicolon-code*) zaczyna się od upewnienia, iż podczas definiowania operatora nie nastąpiła zmiana rozmiaru stosu parametrów. Jeśli tak nie jest, to nastąpi wyprowadzenie komunikatu

DEFINITION NOT FINISHED

a następnie wykonanie czynności przedstawionych łącznie w opisach operatorów ?ERROR i WARNING. W przeciwnym razie nastąpi umieszczenie w słowniku wskazania operatora (;CODE), zanegowanie bitu *smudge* właśnie definiowanego operatora i ustawienie interpretera w stan wykonywania. ;CODE jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ;CODE ?CSP COMPILE (;CODE)
        [COMPILE] [ SMUDGE ; IMMEDIATE
```

;S (--)

Wykonanie operacji ;S (*semicolon*-S) powoduje zakończenie interpretowania operatorów bieżącego poziomu interpretacji. W szczególności wykonanie tej operacji powoduje zakończenie interpretowania definicji utworzonej za pomocą kompilatora : (dwukropek) oraz zakończenie interpretacji ekranów spowodowanej użyciem operatora LOAD. ;S jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

< (*a b* -- *t*)
(*a b* -- *f*)

Wykonanie operacji < (*less-than*) powoduje rozpatrzenie relacji $a<b$. Jeśli relacja ta jest prawdziwa, to na stosie parametrów dane *a* i *b* zostaną zastąpione daną *t* o wartości 1. W przeciwnym razie zostaną zastąpione daną *f* o wartości 0. < jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: < - 0< ;
```

<# (--)

Wykonanie operacji <# (*less-than-sharp*) powoduje wykonanie czynności poprzedzających dokonanie konwersji danej liczbowej na ciąg znaków ASCII, umieszczony następnie w buforze liczby. <# jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: <# PAD HLD ! ;
```

<BUILDS (--)

Wykonanie operacji <BUILDS (*builds*) powoduje utworzenie definicji nowego operatora o nazwie identycznej z nazwą najbliższego słowa

występującego w strumieniu wejściowym. Ponadto w polu parametrów tego operatora zostanie umieszczona dana o wartości 0. Dana ta zostanie zmieniona podczas wykonywania operacji DOES>. <BUILDS jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: <BUILDS 0 CONSTANT ;

= (*a b* -- *t*)
(*a b* -- *f*)

Wykonanie operacji = (*equals*) powoduje rozpatrzenie relacji $a = b$. Jeśli relacja ta jest prawdziwa, to na stosie parametrów dane *a* i *b* zostaną zastąpione daną *t* o wartości 1. W przeciwnym razie zostaną zastąpione daną *f* o wartości 0. = jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: = - 0= ;

> (*a b* -- *t*)
(*a b* -- *f*)

Wykonanie operacji > (*greater-than*) powoduje rozpatrzenie relacji $a > b$. Jeśli relacja ta jest prawdziwa, to na stosie parametrów dane *a* i *b* zostaną zastąpione daną *t* o wartości 1. W przeciwnym razie zostaną zastąpione daną *f* o wartości 0. > jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: > SWAP < ;

>R (*v* --)

Wykonanie operacji >R (*to-R*) powoduje przeniesienie danej słowowej ze stosu parametrów na stos powrotów. >R jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

? (*a* --)

Wykonanie operacji ? (*question-mark*) powoduje wyprowadzenie danej słowowej znajdującej się pod adresem *a*, jako liczby całkowitej z uwzględnieniem bieżącej podstawy liczb. ? jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: ? @ . ;

?COMP (--)

Wykonanie operacji ?COMP (*question-compile*) polega na upewnieniu się, iż interpreter znajduje się w stanie definiowania. Jeśli tak nie jest, to nastąpi wyprowadzenie komunikatu

COMPILATION ONLY, USE IN DEFINITION

i wykonanie czynności przedstawionych łącznie w opisach operatorów

?ERROR i WARNING. ?COMP jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: ?COMP STATE @ 0= 11 ?ERROR ;

?CSP (--)
Wykonanie operacji ?CSP (*question-C-S-P*) powoduje rozpatrzenie adresu szczytu stosu parametrów. Jeśli adres ten jest identyczny z adresem zapamiętanym w zmiennej CSP, to nie stanie się nic. W przeciwnym razie nastąpi wykonanie czynności określonych w opisie zmiennej WARNING, tj. najczęściej wyprowadzenie tekstu
DEFINITION NOT FINISHED
a następnie wykonanie operacji QUIT. ?CSP jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: ?CSP SP@ CSP @ – 14 ?ERROR ;

?ERROR (*b n* --)
Wykonanie operacji ?ERROR (*question-error*) powoduje rozpatrzenie wartości danej *b*. Jeśli dana ta ma wartość 0, to nie stanie się nic. W przeciwnym razie nastąpi wykonanie operacji ERROR, tj. zasygnalizowanie błędu o numerze *n*. Sposób sygnalizowania błędu zależy od bieżącej wartości zmiennej WARNING. Najczęściej sygnalizacja sprowadza się do wyprowadzenia tekstu identyfikowanego przez *n*, a następnie wykonania operacji QUIT. ?ERROR jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: ?ERROR SWAP IF ERROR ELSE DROP THEN ;

?EXEC (--)
Wykonanie operacji ?EXEC (*question-execute*) polega na upewnieniu się, że interpreter znajduje się w stanie wykonywania. Jeśli tak nie jest, to nastąpi wyprowadzenie komunikatu
EXECUTION ONLY
i wykonanie czynności przedstawionych łącznie w opisach operatorów ?ERROR i WARNING.?EXEC jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: ?EXEC STATE @ 12 ?ERROR ;

?LOADING (--)
Wykonanie operacji ?LOADING (*question-loading*) polega na upewnieniu się, iż źródłem strumienia wejściowego jest pamięć dyskowa. Jeśli tak nie jest, to nastąpi wyprowadzenie komunikatu
USE ONLY WHEN LOADING
i wykonanie czynności przedstawionych łącznie w opisach operatorów

?ERROR i WARNING. ?LOADING jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: ?LOADING BLK @ 0= 16 ?ERROR ;

?PAIRS (*a b* --)
Wykonanie operacji ?PAIRS (*question-pairs*) polega na upewnieniu się, iż $a = b$. Jeśli tak nie jest, to nastąpi wyprowadzenie komunikatu
CONDITIONALS NOT PAIRED
i wykonanie czynności przedstawionych łącznie w opisach operatorów ?ERROR i WARNING. ?PAIRS jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: ?PAIRS – 13 ?ERROR ;

?STACK (--)
Wykonanie operacji ?STACK (*question-stack*) powoduje rozpatrzenie stanu stosu parametrów. Jeśli stos ten próbowano skrócić albo wydłużyć ponad miarę, to nastąpi odpowiednio wyprowadzenie komunikatu
EMPTY STACK albo STACK OVERFLOW
i wykonanie czynności przedstawionych łącznie w opisach operatorów ?ERROR i WARNING. ?STACK jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: ?STACK SP@ S0 @ SWAP U< 1 ?ERROR
SP@ HERE 80 + U< 7 ?ERROR ;

?TERMINAL (-- *t*)
(-- *f*)
Wykonanie operacji ?TERMINAL (*question-terminal*) powoduje rozpatrzenie stanu bufora klawiatury terminala. Jeśli bufor ten zawiera jeszcze nie wprowadzony znak, to na stosie parametrów zostanie umieszczona dana *t* o wartości 1. W przeciwnym razie zostanie tam umieszczona dana *f* o wartości 0. ?TERMINAL jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

@ (*a* -- *v*)
Wykonanie operacji @ powoduje zastąpienie adresu *a* pewnej danej słowowej *v* tą właśnie daną. @ jest operatorem biernym zdefiniowanym w kodzie maszynowym.

ABORT (--)
Wykonanie operacji ABORT (*abort*) powoduje przywrócenie stanu początkowego interpretera (*warm start*), tj. wyzerowanie obu stosów, ustawienie interpretera w stan wykonywania i związanie strumienia

wejściowego z klawiaturą terminala. Ponadto zmienne identyfikowane przez operatory CONTEXT i CURRENT otrzymują wartości wyróżniające podsłownik FORTH, a zmienna określająca podstawę liczb otrzymuje wartość 10. ABORT jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ABORT SP! DECIMAL ?STACK CR .CPU
        ." FIG-FORTH 8080 VER 1.1 "
        FORTH DEFINITIONS QUIT ;
```

ABS (*a* -- *b*)

Wykonanie operacji ABS (*absolute*) powoduje zastąpienie — na stosie parametrów — danej *a* daną *b* o wartości bezwzględnej danej *a*. ABS jest operatorem biernym zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ABS DUP +- ;
```

AGAIN (*a n*1 --) por. opis operacji BEGIN

Wykonanie operacji AGAIN (*again*) zaczyna się od upewnienia, iż dana *n*1 ma wartość 1. Jeśli tak nie jest, to operacja AGAIN jest użyta w niepoprawnym kontekście. Spowoduje to wykonanie czynności przedstawionych w opisie operatora ?PAIRS. W przeciwnym razie w słowniku zostanie umieszczone wskazanie operatora BRANCH, a za nim adres względny tego operatora, który został umieszczony w słowniku pod adresem *a*. AGAIN jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: AGAIN 1 ?PAIRS COMPILE BRANCH
        BACK ; IMMEDIATE
```

ALLOT (*n* --)

Wykonanie operacji ALLOT (*allot*) powoduje przemieszczenie wskazania pierwszego wolnego bajtu słownika o *n* bajtów. Dana *n* może być dowolną daną słowową ze znakiem. ALLOT jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ALLOT DP +! ;
```

AND (*a b* -- *c*)

Wykonanie operacji AND (*and*) powoduje zastąpienie — na stosie parametrów — danych słowowych *a* i *b* daną słowową *c* tak dobraną, że stanowi ona rezultat iloczynu logicznego wyznaczonego równolegle na wszystkich parach odpowiadających sobie bitów danych *a* i *b*. AND jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

B/BUF (-- *n*)

Wykonanie operacji B/BUF (*bytes-per-buffer*) powoduje umieszczenie – na stosie parametrów – danej słowowej określającej rozmiar bufora dyskowego, wyrażony w bajtach. Rozmiar ten stanowi jednocześnie jednostkę transmisji danych za pomocą jednokrotnego wykonania operacji BLOCK. Należy nadmienić, że liczba buforów dyskowych może być określona za pomocą sekwencji LIMIT FIRST – B/BUF /. B/BUF jest operatorem biernym, zdefiniowanym za pomocą kompilatora CONSTANT.

B/SCR (-- *n*)

Wykonanie operacji B/SCR (*blocks-per-screen*) powoduje umieszczenie – na stosie parametrów – danej słowowej określającej z jakiej liczby bloków składa się jeden ekran dyskowy. B/SCR jest operatorem biernym, zdefiniowanym za pomocą kompilatora CONSTANT.

BACK (*a* --)

Wykonanie operacji BACK (*back*) powoduje przekształcenie adresu *a* w adres względny, liczony względem adresu pierwszego wolnego bajtu słownika, a następnie umieszczenie tego adresu względnego w słowniku. BACK jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: BACK HERE – , ;

BASE (-- *n*)

Wykonanie operacji BASE (*base*) powoduje umieszczenie – na stosie parametrów – adresu zmiennej słowowej określającej bieżącą podstawę liczb. BASE jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER (wartością początkową zmiennej identyfikowanej przez BASE jest 10).

BEGIN (-- *a* 1)

Wykonanie operacji BEGIN (*begin*) powoduje upewnienie się, że interpreter znajduje się w stanie definiowania. Jeśli tak nie jest, to nastąpi wykonanie czynności przedstawionych w opisie operatora ?COMP. W przeciwnym razie na stosie parametrów zostanie umieszczony adres *a* pierwszego wolnego bajtu słownika oraz dana słowowa o wartości 1. BEGIN jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: BEGIN ?COMP HERE 1 ; IMMEDIATE

BL (— — *n*)

Wykonanie operacji BL (*B-L*) powoduje umieszczenie — na stosie parametrów — danej słowowej *n* o wartości równej kodowi znaku „spacja". BL jest operatorem biernym, zdefiniowanym za pomocą kompilatora CONSTANT.

BLANKS (*a n* — —)

Wykonanie operacji BLANKS (*blanks*) powoduje wypełnienie spacjami obszaru *n* bajtów pamięci, począwszy od adresu *a*. BLANKS jest operatorem biernym zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: BLANKS BL FILL ;
```

BLK (— — *a*)

Wykonanie operacji BLK (*B-L-K*) powoduje umieszczenie — na stosie parametrów — adresu zmiennej słowowej określającej numer tego bloku pamięci dyskowej, z którego pochodzi strumień wejściowy. Jeśli wspomniana zmienna ma wartość 0, to oznacza to, iż strumień wejściowy pochodzi z klawiatury terminala. BLK jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

BLOCK (*n* — — *a*)

Wykonanie operacji BLOCK (*block*) zaczyna się od sprawdzenia, czy w jednym z buforów dyskowych w pamięci operacyjnej znajduje się już blok o numerze względnym *n*. Jeśli go tam nie ma, to zostanie sprowadzony z pamięci dyskowej. Następnie dana słowowa *n*, znajdująca się na stosie parametrów, zostanie zastąpiona adresem *a* trzeciego bajtu tego bufora w pamięci operacyjnej, w którym znajduje się blok dyskowy o numerze *n*. BLOCK jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: BLOCK OFFSET @ + >R PREV @ R – DUP +
          IF
             BEGIN
                +BUF 0=
                IF
                  DROP R BUFFER DUP R 1 R/W 2 –
                THEN
                DUP @ R – DUP + 0=
             UNTIL
             DUP PREV !
          THEN R> DROP 2+ ;
```

BRANCH (--)

Wykonanie operacji BRANCH (*branch*) powoduje dodanie do licznika interpretera IP wartości danej następującej bezpośrednio po wskazaniu operatora BRANCH. Zrealizowanie takiej operacji jest równoważne wykonaniu bezwarunkowego skoku względnego. BRANCH jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

BUFFER (*n* -- *a*)

Wykonanie operacji BUFFER (*buffer*) zaczyna się od zarezerwowania bufora w pamięci operacyjnej dla bloku dyskowego o numerze bezwzględnym *n*. Następnie dana słowowa *n*, znajdująca się na stosie parametrów, zostanie zastąpiona adresem *a* trzeciego bajtu zarezerwowanego bufora. BUFFER jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: BUFFER USE @ DUP >R
         BEGIN
           +BUF
         UNTIL
         USE ! R @ 0<
         IF
           R 2+ R @ 7FFF AND 0 R/W
         THEN
         R ! R PREV ! R> 2+ ;
```

C! (*v a* --)

Wykonanie operacji C! (*C-store*) powoduje umieszczenie pod adresem *a* dolnego bajtu danej słowowej *v*. C! jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

C, (*v* --)

Wykonanie operacji C, (*C-comma*) powoduje przeniesienie do słownika dolnego bajtu danej słowowej *v*. C jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: C, HERE C! 1 ALLOT ;
```

C/L (-- *n*)

Wykonanie operacji C/L (*C-slash-L*) powoduje umieszczenie — na stosie parametrów — danej słowowej *n* określającej rozmiar wiersza ekranu w znakach. C/L jest operatorem biernym, zdefiniowanym za pomocą kompilatora CONSTANT (najczęściej $n = 64$).

C@ (*a* -- *v*)

Wykonanie operacji C@ (*C-fetch*) powoduje zastąpienie — na stosie

parametrów — adresu *a* taką daną *v*, której górny bajt ma wartość 0, a dolny jest identyczny z daną bajtową znajdującą się pod adresem *a*. C@ jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

CFA (*p* — — *c*)
Wykonanie operacji CFA (*C-F-A*) powoduje zastąpienie — na stosie parametrów — adresu *p* pola parametrów operatora adresem *c* pola kodu tego operatora. CFA jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: CFA 2 - ;
```

CMOVE (*s t n* — —)
Wykonanie operacji CMOVE (*C-move*) powoduje przepisanie *n* bajtów, bajt po bajcie, z obszaru pamięci operacyjnej rozpoczynającego się od adresu *s* do obszaru rozpoczynającego się od adresu *t*. CMOVE jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

COLD (— —)
Wykonanie operacji COLD (*cold*) powoduje przywrócenie stanu początkowego interpretera, włącznie ze stanem początkowym zmiennych użytkownika (*cold start*). Powoduje to m.in. przywrócenie początkowego stanu słownika oraz likwidację przyporządkowania buforom pamięci operacyjnej bloków dyskowych. COLD jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: COLD EMPTY-BUFFERS 0 DENSITY ! FIRST USE !
       FIRST PREV ! DR0 0 EPRINT !
       [ 12 +ORIGIN ] LITERAL
       [ 26 +ORIGIN ] LITERAL @ 6 +
       10 CMOVE
       [ 0C +ORIGIN ] LITERAL @
       [ ' FORTH 4 + ] LITERAL !
       ABORT ;
```

Uwaga: przyjęto następujące przyporządkowanie adresów i interpretację umieszczonych pod tymi adresami danych inicjujących (*boot-up literals*):

origin+12 początkowe wartości parametrów S0, R0, TIB, WIDTH, WARNING, FENCE, DP, VOC-LINK
origin+26 adres początkowy obszaru użytkownika (*user area*)
origin+0C adres ostatniej definicji w podsłowniku FORTH

COMPILE (— —)
Wykonanie operacji COMPILE (*compile*) zaczyna się od upewnienia,

że interpreter znajduje się w stanie definiowania. Jeśli tak nie jest, to operacja COMPILE jest użyta w niepoprawnym kontekście. Spowoduje to wyprowadzenie komunikatu

COMPILATION ONLY, USE IN DEFINITION

a następnie wykonanie czynności przedstawionych łącznie w opisach operatorów ?ERROR i WARNING. W przeciwnym razie w słowniku zostanie umieszczone wskazanie, następujące bezpośrednio po właśnie interpretowanym wskazaniu operatora COMPILE. COMPILE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: COMPILE ?COMP >R DUP 2+ R> @ , ;

CONSTANT (*v* --)

Wykonanie operacji CONSTANT (*constant*) powoduje utworzenie definicji operatora o takiej nazwie, jaką ma najbliższe słowo w strumieniu wejściowym. Operator, którego definicję utworzono za pomocą kompilatora CONSTANT, identyfikuje stałą. Oznacza to, że jego wykonanie powoduje umieszczenie — na stosie parametrów — danej określonej podczas jego definiowania. CONSTANT jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: CONSTANT CREATE SMUDGE , ;CODE *kod-maszynowy*

(*kod-maszynowy* reprezentuje program w kodzie maszynowym, którego adres zostanie umieszczony w polu CFA każdego operatora zdefiniowanego za pomocą kompilatora CONSTANT. Wykonanie tego kodu powoduje umieszczenie — na stosie parametrów — stałej identyfikowanej przez operator zdefiniowany za pomocą kompilatora CONSTANT).

CONTEXT (-- *a*)

Wykonanie operacji CONTEXT (*context*) powoduje umieszczenie — na stosie parametrów — adresu zmiennej słowowej, której wartością jest adres słowa należącego do podsłownika, od którego rozpoczyna się wyszukiwanie operatorów. CONTEXT jest operatorem biernym, zdefiniowanym za pomocą kompilatora VARIABLE.

COUNT (*a* -- *b n*)

Wykonanie operacji COUNT (*count*) powoduje zastąpienie — na stosie parametrów — adresu *a* ciągu znakowego rozpoczynającego się od licznika znaków adresem $b = a+1$ pierwszego znaku za licznikiem oraz licznikiem *n*. COUNT jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: COUNT DUP 1+ SWAP C@ ;

CR (--)

Wykonanie operacji CR (*C-R*) powoduje wyprowadzenie pary znaków: powrotu karetki i wysuwu, tj. przemieszczenie kursora na początek nowego wiersza. CR jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

CREATE (--)

Wykonanie operacji CREATE (*create*) powoduje utworzenie nagłówka definicji nowego operatora o nazwie wyrażonej przez najbliższe słowo występujące w strumieniu wejściowym. Nagłówek ten będzie się składać z pola nazwy, pola łącznika i pola kodu. Pole nazwy będzie zawierać ustawiony na wartość 1 bit *smudge*, a pole kodu będzie zawierać adres bajtu następującego bezpośrednio po tym polu. Jeśli w słowniku występuje już operator o podanej nazwie, to utworzenie nowej definicji jest poprzedzone wyprowadzeniem tekstu

ISN'T UNIQUE

Stanowi to ostrzeżenie, iż nowa definicja może przesłonić definicję poprzednią. Ze względu na ustawienie bitu *smudge* nie wyklucza to jednak powołania się w definicji nowej na definicję starą. CREATE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: CREATE -FIND
         IF
           DROP NFA ID. 4 MESSAGE SPACE
         THEN
         HERE DUP C@ WIDTH @ MIN
         1+ ALLOT DUP A0 TOGGLE
         HERE 1 - 80 TOGGLE
         LATEST , CURRENT @ !
         HERE 2+ , ;
```

CSP (-- *a*)

Wykonanie operacji CSP (*C-S-P*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej, której wartością jest zazwyczaj ostatnio zapamiętany adres szczytu stosu parametrów. CSP jest operatorem biernym, zdefiniowanym za pomocą kompilatora VARIABLE.

CURRENT (-- *a*)

Wykonanie operacji CURRENT (*current*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej słowowej, której wartością jest adres słowa, zawierającego adres ostatniego słowa

należącego do podsłownika, w którym są umieszczane definicje nowych operatorów. CURRENT jest operatorem biernym, zdefiniowanym za pomocą kompilatora VARIABLE.

D+ (*da db* — — *ds*)
Wykonanie operacji D+ (*D-plus*) powoduje zastąpienie — na stosie parametrów — danych dwusłowowych *da* i *db* ich dwusłowową sumą *ds* = *da*+*db*. D+ jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

D+ — (*da b* — — *dc*)
Wykonanie operacji D+ — (*D-plus-minus*) powoduje zastąpienie — na stosie parametrów — danej dwusłowowej *da* i danej słowowej *b*, daną dwusłowową *dc*. Jeśli *b* jest ujemne, to *dc* = —*da*. W przeciwnym razie *dc* = *da*. D+ — jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: D+ — 0< IF DMINUS THEN ;

D. (*dv* — —)
Wykonanie operacji D. (*D-dot*) powoduje wyprowadzenie danej *dv*, jako wyrównanej lewostronnie liczby całkowitej, po której następuje jedna spacja. Jeśli wyprowadzana dana ma wartość ujemną, to pierwsza cyfra liczby jest poprzedzona znakiem — (minus). D. jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: D. 0 D.R ;

D.R (*dv w* — —)
Wykonanie operacji D.R (*D-dot-R*) powoduje wyprowadzenie danej dwusłowowej *dv*, jako liczby całkowitej wyrównanej prawostronnie w polu o szerokości *w*. Jeśli wyprowadzana liczba nie mieści się w polu o szerokości *w*, to szerokość pola zostanie niejawnie zwiększona. D.R jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: D.R >R SWAP OVER DABS <# #S SIGN #>
R> OVER SPACES TYPE ;

DABS (*da* — — *db*)
Wykonanie operacji DABS (*D-abs*) powoduje zastąpienie — na stosie parametrów — danej dwusłowowej *da* daną dwusłowową *db* o takiej wartości, jaką ma moduł danej *da*. DABS jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: DABS D+ — ;

DECIMAL (--)

Wykonanie operacji DECIMAL (*decimal*) powoduje nadanie zmiennej BASE wartości 10. Powoduje to, że najbliższe konwersje liczb podczas wprowadzania i wyprowadzania będą wykonywane przy podstawie 10. DECIMAL jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: DECIMAL 0A BASE ! ;
```

DEFINITIONS (--)

Wykonanie operacji DEFINITIONS (*definitions*) powoduje nadanie zmiennej identyfikowanej przez operator CURRENT tej samej wartości jaką ma zmienna identyfikowana przez operator CONTEXT. Ma to taki skutek, że zarówno wyszukiwanie operatorów w słowniku, jak i umieszczanie w nim nowych definicji, będzie dotyczyć tego samego podsłownika. DEFINITIONS jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: DEFINITIONS CONTEXT @ CURRENT ! ;
```

DIGIT (*c b* -- *d t*)
(*c b* -- *f*)

Wykonanie operacji DIGIT (*digit*) powoduje rozstrzygnięcie, czy mniej znaczący bajt danej *c* zawiera znak w kodzie ASCII, będący poprawną cyfrą przy podstawie *b*. W przypadku twierdzącym — na stosie parametrów — dane *c* i *b* zostaną zastąpione daną *d* o wartości liczbowej cyfry i daną *t* o wartości 1. W przypadku przeczącym, dane te zostaną zastąpione daną *f* o wartości 0. DIGIT jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

DLITERAL (*dv* --)

Wykonanie operacji DLITERAL (*D-literal*) powoduje umieszczenie w słowniku takich wskazań operatorów LIT oraz danych, aby podczas przyszłego interpretowania definicji spowodowało to umieszczenie — na stosie parametrów — danej dwusłowowej *dv*. Zinterpretowanie operacji DLITERAL w stanie wykonywania jest równoważne wykonaniu operacji pustej. DLITERAL jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: DLITERAL STATE @
           IF
             SWAP LITERAL LITERAL
           THEN
           ;   IMMEDIATE
```

DMINUS (*da* — — *db*)

Wykonanie operacji DMINUS (*D-minus*) powoduje zastąpienie — na stosie parametrów — danej dwusłowowej *da* taką daną dwusłowową *db*, że *db* = —*da*. DMINUS jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

DO (— — *a* 3)

Wykonanie operacji DO (*do*) zaczyna się od upewnienia, że interpreter znajduje się w stanie definiowania. Jeśli tak nie jest, to nastąpi wykonanie czynności przedstawionych w opisie operatora ?COMP. W przeciwnym razie w słowniku zostanie umieszczone wskazanie operatora (DO), a następnie na stosie parametrów zostanie umieszczony adres pierwszego wolnego bajtu w słowniku oraz dana o wartości 3. DO jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: DO COMPILE (DO) HERE 3 ; IMMEDIATE

DOES> (— —) por. opis operacji <BUILDS

Wykonanie operacji DOES> (*does*) podczas interpretowania definicji pewnego kompilatora powoduje dokonanie zmian w definicji właśnie definiowanego operatora. Zmiany dotyczą pola kodu i pola parametrów. W polu kodu zostanie umieszczony adres programu w kodzie maszynowym, wyszczególnionego w definicji operatora DOES>, a w pierwszym słowie pola parametrów zostanie umieszczony adres bajtu następującego bezpośrednio po wskazaniu omawianego operatora DOES>. DOES> jest operatorem biernym; zdefiniowanym za pomocą kompilatora : (dwukropek)

: DOES> R> LATEST PFA ! ;CODE *kod-maszynowy*

(*kod-maszynowy* reprezentuje program w kodzie maszynowym, którego adres zostanie umieszczony w pierwszym słowie pola parametrów każdego operatora skompilowanego z użyciem konstrukcji <BUILDS ... DOES> lub jej równoważnej).

DP (— — *a*)

Wykonanie operacji DP (*D-P*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej określającej adres pierwszego wolnego bajtu słownika. DP jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

DPL (— — *a*)

Wykonanie operacji DPL (*D-P-L*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej słowowej określającej liczbę cyfr za

ostatnią kropką występującą po cyfrach właśnie wprowadzonej liczby lub jej części. DPL jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

DR0 (--)
Wykonanie operacji DR0 czyni stację numer 0 stacją bieżącą i powoduje nadanie zmiennej identyfikowanej przez operator OFFSET numeru bewzględnego, jaki ma zerowy blok dysku nr 0. DR0 jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: DR0 *dr—0—beg* OFFSET ! ;
(przyjęto, że *dr—0—beg* jest odpowiednio dobraną stałą).

DR1 (--)
Wykonanie operacji DR1 czyni stację numer 0 stacją bieżącą i powoduje nadanie zmiennej identyfikowanej przez operator OFFSET numeru bezwzględnego, jaki ma zerowy blok dysku nr 1. DR0 jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: DR1 *dr—1—beg* OFFSET ! ;
(przyjęto, że *dr—1—beg* jest odpowiednio dobraną stałą).

DROP (*v* --)
Wykonanie operacji DROP (*drop*) powoduje usunięcie — ze stosu parametrów — danej *v*. DROP jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

DUP (*v* -- *v v*)
Wykonanie operacji DUP (*dupe*) powoduje powielenie — na stosie parametrów — danej *v*. DUP jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

ELSE (*a n*2 -- *b* 2) por. opis operacji IF
Wykonanie operacji ELSE (*else*) zaczyna się od upewnienia, że dana *n*2 ma wartość 2. Jeśli tak nie jest, to operacja ELSE jest użyta w niepoprawnym kontekście. Spowoduje to wykonanie czynności przedstawionych w opisie operatora ?PAIRS. W przeciwnym razie w słowniku zostanie umieszczone wskazanie operatora BRANCH oraz dana o wartości zero. Następnie pod adresem *a* zostanie umieszczony adres względny pierwszego wolnego bajtu słownika, a na stosie parametrów pozostaną: adres *b* wspomnianej wyżej danej o wartości 0 oraz dana o wartości 2. ELSE jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ELSE 2 ?PAIRS COMPILE BRANCH HERE 0 ,
        SWAP 2 COMPILE ENDIF 2 ; IMMEDIATE
```

EMIT (*c* --)

Wykonanie operacji EMIT (*emit*) powoduje wyprowadzenie znaku stanowiącego mniej znaczący bajt danej *c*. Ponadto nastąpi zwiększenie o 1 zmiennej identyfikowanej przez operator OUT. EMIT jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: EMIT *pemit* 1 OUT +! ;

(przyjęto, że wykonanie operacji *pemit* spowoduje zdjęcie danej ze stosu parametrów i wyprowadzenie znaku stanowiącego jej mniej znaczący bajt. pemit jest operatorem zdefiniowanym w kodzie maszynowym).

EMPTY-BUFFERS (--)

Wykonanie operacji EMPTY-BUFFERS (*empty-buffers*) powoduje wyzerowanie wszystkich buforów dyskowych w pamięci operacyjnej. Powoduje to m.in. ustawienie na wartość 0 wszystkich bitów update tych buforów. EMPTY-BUFFERS jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: EMPTY—BUFFERS FIRST LIMIT OVER — ERASE ;

ENCLOSE (*a c* -- *a n*1 *n*2 *n*3)

Wykonanie operacji ENCLOSE (*enclose*) powoduje rozpatrzenie ciągu znaków znajdującego się w pamięci operacyjnej pod adresem *a*. Ciąg ten jest analizowany bajt po bajcie i porównywany z dolnym bajtem danej *c*, który jest uznawany za ogranicznik. W następstwie tego porównania dana *c* zostanie zastąpiona danymi *n*1, *n*2 i *n*3 stanowiącymi adresy względne (liczone względem adresu *a*) o następującej interpretacji:

*n*1 — adres względny pierwszego znaku, który nie jest ogranicznikiem,

*n*2 — adres względny pierwszego ogranicznika następującego po ciągu znaków, z których żaden nie jest ogranicznikiem,

*n*3 — adres względny pierwszego znaku, którego jeszcze nie rozpatrywano.

ENCLOSE jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

END (*a n* --)

Wykonanie operacji END (*end*) jest równoważne wykonaniu operacji UNTIL. END jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: END [COMPILE] UNTIL ; IMMEDIATE

ENDIF (*a n*2 --) por. opis operacji IF, ELSE

Wykonanie operacji ENDIF (*endif*) zaczyna się od upewnienia, że dana *n*2 ma wartość 2. Jeśli tak nie jest, to operacja ENDIF jest użyta w niepoprawnym kontekście. Spowoduje to wykonanie czynności przedstawionych w opisie operatora ?PAIRS. W przeciwnym razie słowo występujące pod adresem *a* zostanie zastąpione adresem względnym pierwszego wolnego bajtu w słowniku. ENDIF jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ENDIF ?COMP 2 ?PAIRS HERE OVER -
        SWAP ! ; IMMEDIATE
```

ERASE (*a n* --)

Wykonanie operacji ERASE (*erase*) powoduje wypełnienie bajtami o wartości 0 pola *n* bajtów zaczynającego się od adresu *a*. ERASE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ERASE 0 FILL ;
```

ERROR (*n* -- *i b*)

Wykonanie operacji ERROR (*error*) ma za zadanie wyprowadzenie komunikatu o zaistniałym błędzie nr *n*, jednak sposób zrealizowania tego zadania zależy od bieżącej wartości zmiennej WARNING. Jeśli WARNING ma wartość ujemną, to poza wyprowadzeniem komunikatu nastąpi wykonanie operacji (ABORT). W przeciwnym razie, zamiast niej zostanie wykonana operacja QUIT. W tym drugim przypadku dana *n* zostanie zastąpiona — na stosie parametrów — — parą danych *i* oraz *b*, określających odpowiednio bieżące wartości zmiennych identyfikowanych przez operatory IN i BLK. Dane te umożliwiają bliższą identyfikację miejsca i przyczyny zaistnienia błędu. W typowych implementacjach przewidziano następujące komunikaty o błędach:

0 –
brak definicji operatora
1 EMPTY STACK
pusty stos parametrów
2 DICTIONARY FULL
brak miejsca w słowniku
3 HAS INCORRECT ADDRESS MODE
niepoprawny tryb adresowania
4 ISN'T UNIQUE
redefinicja operatora
6 DISC RANGE?

operacja dotycząca nieistniejącego bloku

7 FULL STACK
przepełnienie stosu

8 DISC ERROR
błąd współpracy z dyskiem

15 FORTH INTEREST GROUP
komunikat podczas wykonywania operacji TRIAD

17 COMPILATION ONLY, USE IN DEFINITION
próba zinterpretowania operacji dopuszczalnej jedynie w stanie definiowania

18 EXECUTION ONLY
próba zinterpretowania operacji dopuszczalnej jedynie w stanie wykonywania

19 CONDITIONALS NOT PAIRED
błędnie utworzona instrukcja strukturalna

20 DEFINITION NOT FINISHED
zmiana rozmiaru stosu parametrów podczas definiowania operatora

21 IN PROTECTED DICTIONARY
naruszenie ograniczeń narzuconych przez wykonanie operacji FENCE

22 USE ONLY WHEN LOADING
próba wykonania operacji dopuszczalnej jedynie podczas interpretowania strumienia wejściowego pochodzącego z dysku

23 OFF CURRENT EDITING SCREEN
sygnalizacja edytora

24 DECLARE VOCABULARY
próba wykonania operacji FORGET w sytuacji, gdy zmienne identyfikowane przez operatory CONTEXT i CURRENT mają różne wartości.

ERROR jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ERROR WARNING @ 0
        IF (ABORT) THEN
        HERE COUNT TYPE ."? MESSAGE SP!
        IN @ BLK @ QUIT ;
```

EXECUTE (*a* — —)

Wykonanie operacji EXECUTE (*execute*) powoduje wykonanie operacji, której pole kodu znajduje się pod adresem *a*. EXECUTE jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

EXPECT (*a n* --)

Wykonanie operacji EXPECT (*expect*) powoduje wprowadzenie z klawiatury terminala nie więcej niż *n* znaków i umieszczenie ich w polu pamięci operacyjnej, począwszy od adresu *a*. Wprowadzanie jest kończone na podstawie licznika znaków albo po napotkaniu znaku cr (powrót karetki). Po ostatnim wprowadzonym znaku są umieszczane znaki o kodzie 0. EXPECT jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: EXPECT OVER + OVER
         DO
            KEY DUP 0E +ORIGIN @ =
            IF
              DROP 08 OVER I = DUP
              R> 2 - + >R -
            ELSE
              DUP 0D =
              IF
                LEAVE DROP BL 0
              ELSE
                DUP
              THEN
              I  C! 0 I 1+ !
            THEN
            EMIT
         LOOP
         DROP ;
```

FENCE (-- *a*)

Wykonanie operacji FENCE (*fence*) powoduje zabezpieczenie przed usunięciem za pomocą operacji FORGET, tych definicji operatorów, które zajmują w słowniku miejsce poniżej adresu *a*. FENCE jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

FILL (*a n c* --)

Wykonanie operacji FILL (*fill*) powoduje umieszczenie w polu pamięci operacyjnej, zaczynającym się od adresu *a*, *n* znaków identycznych ze znakiem reprezentowanym w dolnym bajcie danej *c*. FILL jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

FIRST (-- *a*)

Wykonanie operacji FIRST (*first*) powoduje umieszczenie — na stosie parametrów — adresu *a* pierwszego bajtu pierwszego bufora dyskowego znajdującego się w pamięci operacyjnej. FIRST jest operatorem biernym, zdefiniowanym za pomocą kompilatora CONSTANT.

FLD (-- *a*)

Wykonanie operacji FLD (*F-L-D*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej określającej rozmiar pola zewnętrznego zajmowanego przez liczbę wyprowadzaną za pomocą wzorców. FLD jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER. Operator FLD wychodzi z użycia i nie powinien być już używany.

FLUSH (--)

Wykonanie operacji FLUSH (*flush*) powoduje wyprowadzenie do pamięci dyskowej tych wszystkich buforów znajdujących się w pamięci operacyjnej, które mają ustawiony bit *update*. FLUSH jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: FLUSH NBUF 1+ 0
        DO
          0 BUFFER DROP
        LOOP ;
```

FORGET (--)

Wykonanie operacji FORGET (*forget*) powoduje usunięcie (ze słownika) definicji operatora o nazwie identycznej z nazwą najbliższego słowa występującego w strumieniu wejściowym, jak również usunięcie wszystkich definicji następujących po niej. W celu zabezpieczenia się przed omyłkowym usunięciem definicji pewnych operatorów brana jest pod uwagę bieżąca wartość zmiennej FENCE. FORGET jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: FORGET CURRENT @ CONTEXT @ - 18 ?ERROR
         [COMPILE] ' DUP FENCE @ < 15 ?ERROR
         DUP NFA DP ! LFA @ CURRENT @ ! ;
```

Przedstawiona definicja operatora FORGET ma tę poważną wadę, że nie uwzględnia podziału słownika na podsłowniki, co może powodować, iż użycie jej w złożonym kontekście spowoduje niekontrolowane załamanie się interpretacji. Dla porównania można za [7] podać definicję operatora FORGET nie obarczoną tym zagrożeniem

```
: FORGET [COMPILE] ' NFA DUP FENCE @ U<
         15 ?ERROR >R VOC-LINK @
         BEGIN
           R OVER U< WHILE
              [COMPILE] FORTH DEFINITIONS @
         REPEAT
         DUP VOC-LINK !
         BEGIN
           DUP 4 -
```

```
            BEGIN
              PFA LFA @ DUP R U<
            UNTIL
            OVER 2 - ! @ -DUP 0=
          UNTIL
          R> DP ! ;
```

FORTH (--)

Wykonanie operacji FORTH (*forth*) powoduje, że bieżącym podsłownikiem wyszukiwania staje się podsłownik FORTH. FORTH jest operatorem czynnym, zdefiniowanym za pomocą kompilatora VOCABULARY

VOCABULARY FORTH IMMEDIATE

HERE (-- *a*)

Wykonanie operacji HERE (*here*) powoduje umieszczenie – na stosie parametrów – adresu *a* pierwszego wolnego bajtu w słowniku. HERE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: HERE DP @ ;

HEX (--)

Wykonanie operacji HEX (*hex*) powoduje nadanie zmiennej identyfikowanej przez operator wartości 16. Powoduje to, że najbliższe konwersje liczb podczas wprowadzania i wyprowadzania będą wykonywane przy podstawie 16. HEX jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: HEX 10 BASE ! ;

HLD (-- *a*)

Wykonanie operacji HLD (*H-L-D*) powoduje umieszczenie – na stosie parametrów – adresu zmiennej słowowej określającej adres ostatniego znaku umieszczonego w buforze liczby. HLD jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

HOLD (*c* --)

Wykonanie operacji HOLD (*hold*) powoduje umieszczenie znaku reprezentowanego w dolnym bajcie danej *c*, w kolejnym bajcie pola liczby. HOLD jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: HOLD -1 HLD +! HLD @ C! ;

I (-- *v*) por. opis operacji (DO)

Wykonanie operacji I (*I*) powoduje umieszczenie — na stosie parametrów — danej *v* o wartości równej bieżącej wartości zmiennej sterującej najwęższego cyklu zainicjowanego w następstwie wykonania operacji (DO), a obejmującego rozpatrywane odwołanie do operatora I. I jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

ID. (*a* --)

Wykonanie operacji ID. (*I-D-dot*) powoduje wyprowadzenie nazwy (zakończonej spacją) tego operatora, którego pole nazwy rozpoczyna się od adresu *a*. Jeśli licznik znaków nazwy operatora, występujący w polu nazwy, ma wartość większą niż bieżąca wartość zmiennej WIDTH, to po obciętej nazwie występują dodatkowe spacje. ID. jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ID. PAD 20 5F FILL DUP PFA LFA OVER -
    PAD SWAP CMOVE PAD COUNT 1F AND
    TYPE SPACE . ;
```

IF (-- *a* 2)

Wykonanie operacji IF (*if*) powoduje umieszczenie w słowniku wskazania operatora 0BRANCH, a za nim danej słowowej o wartości 0 (dana ta zostanie zmieniona podczas interpretowania związanej z IF operacji ELSE, THEN albo ENDIF). Na stosie parametrów zostanie umieszczony adres *a*, wspomnianej danej o wartości 0, a następnie dana o wartości 2. IF jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: IF COMPILE 0BRANCH HERE 0 ,
    2 ; IMMEDIATE
```

IMMEDIATE (--)

Wykonanie operacji IMMEDIATE (*immediate*) powoduje zanegowanie bitu *immediate* występującego w polu nazwy tego operatora, który jest zdefiniowany jako ostatni w podsłowniku określonym przez zmienną identyfikowaną przez operator CURRENT. IMMEDIATE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: IMMEDIATE LATEST 40 TOGGLE ;
```

IN (-- *a*)

Wykonanie operacji IN (*in*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej określającej adres względny bajtu wyznaczony względem początku tego bufora, z którego pochodzi strumień wejściowy. Buforem takim może być bufor dyskowy albo

bufor tekstu. Bezpośrednio po wykonaniu operacji QUERY, LOAD oraz --> rozpatrywany adres względny ma wartość 0. IN jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

INDEX (*a b* --)

Wykonanie operacji INDEX (*index*) powoduje wyprowadzenie pierwszych wierszy ekranów o numerach od *a* do *b*. Wykonywanie tej operacji może zostać przerwane przez naciśnięcie dowolnego klawisza klawiatury. INDEX jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: INDEX 0C EMIT CR 1+ SWAP
  DO
     CR I 3 .R SPACE 0 I .LINE
     ?TERMINAL IF LEAVE THEN
  LOOP ;
```

INTERPRET (--)

Wykonanie operacji INTERPRET (*interpret*) powoduje podjęcie interpretacji strumienia wejściowego. Jeśli wartością zmiennej identyfikowanej przez operator BLK jest 0, to strumień wejściowy pochodzi z klawiatury. W przeciwnym razie pochodzi z bloku dyskowego o numerze określonym przez wspomnianą zmienną. INTERPRET jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: INTERPRET BEGIN
             -FIND
             IF
               STATE @ <
               IF
                 CFA ,
               ELSE
                 CFA EXECUTE
               THEN
               ?STACK
            ELSE
               HERE NUMBER DPL @ 1+
               IF
                  [COMPILE] DLITERAL
               ELSE
                  DROP [COMPILE] LITERAL
               THEN
               ?STACK
            THEN
           AGAIN ;
```

KEY (-- *v*)
Wykonanie operacji KEY (*key*) powoduje wprowadzenie z klawiatury terminala jednego znaku i umieszczenie — na stosie parametrów — takiej danej, której bardziej znaczący bajt ma wartość 0, a mniej znaczący stanowi reprezentację wprowadzonego znaku. KEY jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

LATEST (-- *a*)
Wykonanie operacji LATEST (*latest*) powoduje umieszczenie — na stosie parametrów — adresu *a* pola nazwy tego operatora, który jest zdefiniowany jako ostatni w podsłowniku określonym przez zmienną CURRENT. LATEST jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: LATEST CURRENT @ @ ;
```

LEAVE (--) por. opis operacji (DO)
Wykonanie operacji LEAVE (*leave*) powoduje taką zmianę parametrów cyklu zainicjowanego w następstwie wykonania operacji (DO), aby podczas najbliższego rozstrzygania o kontynuowaniu cyklu nastąpiło jego zakończenie. Wykonanie operacji LEAVE nie ma wpływu na wartość zmiennej sterującej cyklu. LEAVE jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

LFA (*p* -- *c*)
Wykonanie operacji LFA (*L-F-A*) powoduje zastąpienie — na stosie parametrów — adresu *p* pola parametrów operatora adresem *c* pola kodu tego operatora. LFA jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: LFA 04 - ;
```

LIMIT (-- *a*)
Wykonanie operacji LIMIT (*limit*) powoduje umieszczenie — na stosie parametrów — adresu *a* pierwszego bajtu następującego po ostatnim buforze dyskowym w pamięci operacyjnej. LIMIT jest operatorem biernym, zdefiniowanym za pomocą kompilatora CONSTANT.

LIST (*n* --)
Wykonanie operacji LIST (*list*) powoduje wyprowadzenie zawartości ekranu o numerze *n*. LIST jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: LIST DECIMAL CR DUP SCR !
        ." SCR = . " 10 0
        DO
```

```
        CR I 3 .R SPACE I SCR @ .LINE
      LOOP
      CR ;
```

LIT (-- *v*)

Wykonanie operacji LIT (*lit*) powoduje umieszczenie — na stosie parametrów — danej *v* identycznej z daną następującą po wskazaniu operatora LIT. LIT jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

LITERAL (*v* --)

Wykonanie operacji LITERAL (*literal*) powoduje umieszczenie w słowniku wskazania operatora LIT, a bezpośrednio za nim danej słowowej *v*. LITERAL jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: LITERAL STATE @
          IF COMPILE LIT , THEN
          ; IMMEDIATE
```

LOAD (*n* --)

Wykonanie operacji LOAD (*load*) powoduje zinterpretowanie ekranu o numerze *n*. Jeśli ostatnim operatorem takiego ekranu jest -->, to nastąpi także zinterpretowanie ekranu *n*+1 itd. LOAD jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: LOAD BLK @ >R IN @ >R 0 IN !
       B/SCR * BLK ! INTERPRET
       R> IN ! R> BLK ! ;
```

LOOP (*a n*3 --)

Wykonanie operacji LOOP (*loop*) zaczyna się od upewnienia, że dana *n*3 ma wartość 3. Jeśli tak nie jest, to operacja LOOP jest użyta w niepoprawnym kontekście. Spowoduje to wykonanie czynności przedstawionych w opisie operatora ?PAIRS. W przeciwnym razie w słowniku zostanie umieszczone wskazanie operatora (LOOP), a za nim adres względny tego operatora, który został umieszczony w słowniku pod adresem *a*. LOOP jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: LOOP 3 ?PAIRS COMPILE (LOOP)
       BACK ; IMMEDIATE
```

M* (*a b* -- *dv*)

Wykonanie operacji M* (*M-star*) powoduje zastąpienie — na stosie parametrów — danych słowowych *a* i *b* ich dwusłowowym iloczynem

dv. M* jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: M* OVER OVER XOR >R ABS
     SWAP ABS U* R> D+- ;
```

M/ (*dv a* -- *r q*)

Wykonanie operacji M/ (*M-slash*) powoduje zastąpienie – na stosie parametrów – danej dwusłowowej *dv* i danej słowowej *a* parą danych *r* i *q*, tak dobranych, że $q = dv/a$, zaś *r* jest resztą z dzielenia *dv* przez *a*. M/ jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: M/ OVER >R >R DABS R ABS U/
     R> R XOR +- SWAP R> +- SWAP ;
```

M/MOD (*dv a* -- *r dq*)

Wykonanie operacji M/MOD (*M-slash-mod*) powoduje zastąpienie – na stosie parametrów – danej dwusłowowej bez znaku *dv* i danej słowowej bez znaku *a* parą danych bez znaku *r* i *dq*, tak dobranych, że $dq = dv / a$, zaś *r* jest resztą z dzielenia *dv* przez *a*. M/MOD jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: M/MOD >R 0 R U/ R> SWAP >R U/ R> ;
```

MAX (*a b* -- *v*)

Wykonanie operacji MAX (*max*) powoduje zastąpienie – na stosie parametrów – danych *a* i *b* większą z tych danych *v*. MAX jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: MAX OVER OVER <
      IF SWAP THEN
      DROP ;
```

MESSAGE (*n* --)

Wykonanie operacji MESSAGE (*message*) powoduje wyprowadzenie komunikatu o numerze *n*. Postać komunikatu zależy od bieżącej wartości zmiennej identyfikowanej przez operator WARNING. Jeśli zmienna ta ma wartość 0, to komunikat ogranicza się do podania numeru *n*. W przeciwnym razie komunikat ma charakter opisowy i pochodzi z ekranów 4 i 5 stacji dyskowej nr 0. MESSAGE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: MESSAGE  WARNING @
           IF
             -DUP
             IF
               4 OFFSET @ B/SCR / - .LINE
             THEN
           ELSE
             ." MSG" .
           THEN ;
```

MIN (*ab* − −*v*)
Wykonanie operacji MIN (min) powoduje zastąpienie — na stosie parametrów — danych *a* i *b* mniejszą z tych danych *v*. MIN jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: MIN OVER OVER >
      IF SWAP THEN
      DROP ;
```

MINUS (*a* − − *b*)
Wykonanie operacji MINUS (*minus*) powoduje zastąpienie — na stosie paramerów — danej *a* daną $b = -a$. MINUS jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

MOD (*a b* − − *r*)
Wykonanie operacji MOD (*mod*) powoduje zastąpienie — na stosie parametrów — danych *a* i *b* daną *r* o wartości równej reszcie z dzielenia *a* przez *b*. MOD jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: MOD /MOD DROP ;
```

MON (− −)
Wykonanie operacji MON (*mon*) powoduje zakończenie interpretacji i powrót do systemu operacyjnego. W systemie CP/M operacja MON występuje pod nazwą BYE. MON jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

NFA (*p* − − *n*)
Wykonanie operacji NFA (*N-F-A*) powoduje zastąpienie — na stosie parametrów — adresu *p* pola parametrów operatora, adresem *n* pola nazwy tego operatora. NFA jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: NFA 5 - -1 TRAVERSE ;
```

NUMBER (*a* -- *dv*)

Wykonanie operacji NUMBER (*number*) powoduje konwersję liczby, zapisanej za pomocą ciągu znaków kodu ASCII, na daną dwusłowową *dv*. Przyjmuje się, że rozpatrywany ciąg znaków jest umieszczony w pamięci operacyjnej począwszy od adresu *a* i rozpoczyna się od bajtu określającego długość ciągu poddawanego konwersji. Jeśli w rozpatrywanym ciągu znaków występują znaki . (kropka), to pozycja ostatniego z takich znaków zostanie zapamiętana w zmiennej identyfikowanej przez operator DPL. Jeśli wykonanie konwersji okaże się niemożliwe, to zostanie wyprowadzony komunikat składający się ze znaku zapytania. NUMBER jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: NUMBER  0 0 ROT DUP 1+ C@ 2D =
          DUP >R + -1
          BEGIN
            DPL ! (NUMBER) DUP C@ BL -
            WHILE
              DUP C@ 2E - 0 ?ERROR 0
          REPEAT
          DROP R>
          IF
            MINUS
          THEN ;
```

OFFSET (-- *a*)

Wykonanie operacji OFFSET (*offset*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej określającej bezwzględny numer zerowego bloku bieżącej stacji dyskowej. OFFSET jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

OR (*a b* -- *c*)

Wykonanie operacji OR (*or*) powoduje zastąpienie — na stosie parametrów — danych *a* i *b* ich sumą logiczną *c*, wyznaczoną równolegle na wszystkich odpowiadających sobie parach bitów danych *a* i *b*. OR jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

OUT (-- *a*)

Wykonanie operacji OUT (*out*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej słowowej zawierającej licznik znaków zwiększany każdorazowo o 1 podczas wykonania operacji EMIT. OUT jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

OVER (*a b* –– *a b a*)

Wykonanie operacji OVER (*over*) powoduje umieszczenie – na stosie parametrów – takiej samej danej słowowej, jaka poprzedza daną słowową znajdującą się na szczycie stosu. OVER jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

PAD (–– *a*)

Wykonanie operacji PAD (*pad*) powoduje umieszczenie – na stosie parametrów – adresu bufora tekstu. Należy nadmienić, że bufor ten nie zajmuje w pamięci operacyjnej stałego miejsca. Znajduje się on w ustalonej odległości od zmieniającego swoje położenie końca słownika. PAD jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: PAD HERE 44 + ;
```

PFA (*n* –– *p*)

Wykonanie operacji PFA (*P-F-A*) powoduje zastąpienie – na stosie parametrów – adresu *n* pola nazwy operatora adresem *p* jego pola parametrów. PFA jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: PFA 1 TRAVERSE 5 + ;
```

PREV (–– *a*)

Wykonanie operacji PREV (*prev*) powoduje umieszczenie – na stosie parametrów – adresu *a* zmiennej słowowej określającej adres ostatnio użytego bufora dyskowego w pamięci operacyjnej. Początkową wartością tej zmiennej jest adres pierwszego z takich buforów. PREV jest operatorem biernym, zdefiniowanym za pomocą kompilatora VARIABLE.

QUERY (––)

Wykonanie operacji QUERY (*query*) powoduje wprowadzenie z klawiatury, do bufora terminala, nie więcej niż 80 znaków. Wprowadzanie jest kończone na podstawie licznika znaków albo po rozpoznaniu znaku cr (powrót karetki). Znak cr nie jest umieszczany w buforze terminala. Po wprowadzonych znakach w buforze tym jest umieszczany znak o kodzie 0 oraz spacja. Po wykonaniu tych czynności zmienna identyfikowana przez operator IN otrzymuje wartość 0. QUERY jest operatorem biernym, który może być użyty jedynie w stanie wykonywania. Jest on zdefiniowany za pomocą kompilatora : (dwukropek)

```
: QUERY TIB @ 50 EXPECT 0 IN ! ;
```

QUIT (--)

Wykonanie operacji QUIT (*quit*) powoduje zaniechanie interpretowania bieżącego strumienia wejściowego, wyzerowanie stosu powrotów i podjęcie interpretacji strumienia wejściowego związanego z klawiaturą terminala. QUIT jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: QUIT  0 BLK ! [COMPILE] [
        BEGIN
           RP! CR QUERY INTERPRET STATE @ 0=
           IF ." OK" THEN
        AGAIN ;
```

R (-- *v*)

Wykonanie operacji R (*R*) powoduje umieszczenie – na stosie parametrów – takiej samej danej słowowej *v*, jaka występuje na szczycie stosu powrotów. R jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: R [ ' I -2 ALLOT , ] ; -2 ALLOT
```

(tzn. wykonanie operacji R ma taki sam skutek jak wykonanie operacji I).

R# (-- *a*)

Wykonanie operacji R# (*R-hash*) powoduje umieszczenie – na stosie parametrów – adresu zmiennej *a* określającej adres kursora wykorzystywanego podczas edycji ekranu. R# jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

R/W (*a n b* --)

Wykonanie operacji R/W (*R-slash-W*) zaczyna się od rozpatrzenia wartości danej *b*. Jeśli dana ta ma wartość 0, to będzie realizowane wyprowadzanie danych. Jeśli ma wartość różną od zera, to będzie realizowane wprowadzanie. Właściwa transmisja polega na przeniesieniu danych między buforem pamięci dyskowej, znajdującym się w pamięci operacyjnej pod adresem *a*, a blokiem w pamięci zewnętrznej o numerze względnym *n*. Sposób wykonania operacji R/W zależy od właściwości użytego systemu operacyjnego. R/W jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: R/W  USE >R SWAP SEC/BLK *
       ROT USE ! SEC/BLK 0
       DO
          OVER OVER ts-cal
          IF
            secrd
```

```
        ELSE
          secrd
        THEN
        1+ 80 USE +!
     LOOP
     DROP DROP R> USE ! ;
```

(Przedstawiona implementacja dotyczy środowiska CP/M. Przyjęto, że operacja *ts-cal* realizuje przekształcenie bezwzględnego adresu bloku w adres sprzętowy, zaś *secrd* i *secrw* realizują odpowiednio wprowadzenie i wyprowadzenie jednego sektora.)

R0 (-- *a*)

Wykonanie operacji R0 (*R-zero*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej słowowej określającej początkowy adres szczytu stosu powrotów. R0 jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

R> (-- *v*)

Wykonanie operacji R> (*R-from*) powoduje zdjęcie danej słowowej *v* ze stosu powrotów i umieszczenie jej na stosie parametrów. R> jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

REPEAT (*a n*1 *b n*4 --) por. opis operacji BEGIN

Wykonanie operacji REPEAT (*repeat*) zaczyna się od upewnienia, że dana *n*1 ma wartość 1. Jeśli tak nie jest, to operacja REPEAT jest użyta w niepoprawnym kontekście. Spowoduje to wykonanie czynności przedstawionych w opisie operatora ?PAIRS. W przeciwnym razie w słowniku zostanie umieszczone wskazanie operatora BRANCH, a za nim adres względny tego operatora, który został umieszczony w słowniku pod adresem *a*. Następnie odbywa się sprawdzenie, że dana *n*4 ma wartość 4. Jeśli tak nie jest, to operacja REPEAT jest użyta w niepoprawnym kontekście, co jak już omawiano, spowoduje wykonanie czynności przedstawionych w opisie operatora ?PAIRS. W przeciwnym razie pod adresem *b* zostanie umieszczony adres względny pierwszego wolnego bajtu słownika. REPEAT jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: REPEAT >R >R [COMPILE] AGAIN R> R> 2 -
          [COMPILE] ENDIF ; IMMEDIATE
```

ROT (*a b c* -- *b c a*)

Wykonanie operacji ROT (*rote*) powoduje zastąpienie — na stosie parametrów — danych *a*, *b*, *c* danymi *b*, *c*, *a*. ROT jest operatorem

biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: ROT >R SWAP >R SWAP ;

RP! (--)
Wykonanie operacji RP! (*R-P-store*) powoduje wyzerowanie stosu powrotów. Polega to na przypisaniu zmiennej, wskazującej szczyt tego stosu, wartości określonej przez zmienną identyfikowaną przez operator R0. RP! jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

RP@ (-- *a*)
Wykonanie operacji RP@ (*R-P-at*) powoduje umieszczenie — na stosie parametrów — bieżącego adresu *a* szczytu stosu powrotów. RP@ jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

S ->D (*a* -- *db*)
Wykonanie operacji S ->D (*single-to-double*) powoduje zastąpienie — na stosie parametrów — danej słowowej *a* daną słowową *db* o tej samej wartości liczbowej. S ->D jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

S0 (-- *a*)
Wykonanie operacji S0 (*S-zero*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej określającej początkowy adres szczytu stosu parametrów. S0 jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

SCR (-- *a*)
Wykonanie operacji SCR (*S-C-R*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej słowowej określającej numer względny ekranu, którego dotyczyła ostatnio wykonana operacja LIST. LIST jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

SIGN (*b dv* -- *dv*)
Wykonanie operacji SIGN (*sign*) powoduje zastąpienie — na stosie parametrów — danej słowowej *b* i danej dwusłowowej *dv* daną dwusłowową *dv*. Ponadto, jeśli dana *b* ma wartość ujemną, w buforze liczby zostanie umieszczony znak — (minus). SIGN jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: SIGN ROT 0< IF 2D HOLD THEN ;

SMUDGE (--)
Wykonanie operacji SMUDGE (*smudge*) powoduje zanegowanie bitu

smudge w polu nazwy tego operatora, który jest zdefiniowany jako ostatni w podsłowniku określonym przez zmienną identyfikowaną przez operator CURRENT. SMUDGE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: SMUDGE LATEST 20 TOGGLE ;
```

SP! (--)

Wykonanie operacji SP! (*S-P-store*) powoduje wyzerowanie stosu parametrów. Polega to na przypisaniu zmiennej wskazującej szczyt stosu wartości określonej przez zmienną identyfikowaną przez operator S0. SP! jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

SP@ (-- *a*)

Wykonanie operacji SP@ (*S-P-fetch*) powoduje umieszczenie — na stosie parametrów — adresu szczytu stosu parametrów tuż przed wykonaniem tej operacji. SP@ jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

SPACE (--)

Wykonanie operacji SPACE (*space*) powoduje wyprowadzenie jednej spacji. SPACE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: SPACE BL EMIT ;
```

SPACES (*n* --)

Wykonanie operacji SPACES (*spaces*) powoduje wyprowadzenie *n* spacji. SPACES jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: SPACES 0 MAX -DUP
          IF
            0 DO SPACE LOOP
          THEN ;
```

STATE (-- *a*)

Wykonanie operacji STATE (*state*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej określającej stan interpretera. Jeśli zmienna ta ma wartość 0, to interpreter znajduje się w stanie wykonywania. W przeciwnym razie interpreter znajduje się w stanie definiowania. STATE jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

SWAP (*a b* -- *b a*)

Wykonanie operacji SWAP (*swap*) powoduje zastąpienie — na stosie

operatorów — danych *a* i *b* danymi *b* i *a*. SWAP jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

TASK (— —)
Wykonanie operacji TASK (*task*) nie ma żadnych skutków. Definicja tej operacji jest zazwyczaj umieszczana w słowniku przed wszystkimi innymi definicjami. Umożliwia to użycie jej jako argumentu operacji FORGET dla przywrócenia początkowego stanu słownika. TASK jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: TASK ;

THEN (*b n*2 — —) por. opis operacji IF
Wykonanie operacji THEN (*then*) zaczyna się od upewnienia, że dana *n*2 ma wartość 2. Jeśli tak nie jest, to operacja THEN jest użyta w niepoprawnym kontekście. Spowoduje to wykonanie czynności przedstawionych w opisie operatora ?PAIRS. W przeciwnym razie pod adresem *b* zostanie umieszczony adres względny pierwszego wolnego bajtu słownika. THEN jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: THEN [COMPILE] ENDIF ; IMMEDIATE

TIB (— — *a*)
Wykonanie operacji TIB (*T-I-B*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej słowowej określającej adres bufora terminala. TIB jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

TOGGLE (*a b* — —)
Wykonanie operacji TOGGLE (*toggle*) powoduje zastąpienie danej słowowej znajdującej się pod adresem *a* daną słowową stanowiącą różnicę symetryczną tej danej i danej słowowej *b*. Różnica symetryczna jest wyznaczana równolegle na wszystkich parach bitów argumentów tej operacji. TOGGLE jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

TRAVERSE (*a b* — — *c*)
Wykonanie operacji TRAVERSE (*traverse*) powoduje zastąpienie — na stosie parametrów — adresu *a* skrajnego bajtu pola nazwy i danej słowowej *b* adresem drugiego skrajnego bajtu pola nazwy. Jeśli dana *b* ma wartość +1, to przeglądanie pola nazwy odbywa się w kierunku adresów wyższych, a jeśli ma wartość −1, to odbywa się w kierunku adresów niższych. TRAVERSE jest operatorem biernym,

zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: TRAVERSE SWAP
           BEGIN
             OVER + 7F OVER C@
           UNTIL
           SWAP DROP ;
```

TRIAD (*n* --)

Wykonanie operacji TRIAD (*triad*) powoduje wyprowadzenie zawartości trzech ekranów. Wyprowadzanie rozpoczyna się od ekranu, którego numer jest podzielny przez 3. Jednym z wyprowadzanych ekranów jest ekran o numerze *n*. TRIAD jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: TRIAD 0C EMIT 3 / 3 * 3 OVER + SWAP
        DO
          CR I LIST ?TERMINAL
          IF LEAVE THEN
        LOOP
        CR OF MESSAGE CR ;
```

TYPE (*a n* --)

Wykonanie operacji TYPE (*type*) powoduje wyprowadzenie *n* znaków z pola pamięci operacyjnej rozpoczynającego się od adresu *a*. TYPE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: TYPE -DUP
       IF
         OVER + SWAP
         DO
           I C@ EMIT
         LOOP
       ELSE
         DROP
       THEN ;
```

U< (*a b* -- *t*)
(*a b* -- *f*)

Wykonanie operacji U< (*U-less*) powoduje rozpatrzenie relacji $a < b$ dwóch danych słowowych bez znaku. Jeśli relacja ta okaże się prawdziwa, to na stosie parametrów dane te zostaną zastąpione daną *t* o wartości 1. W przeciwnym razie zostaną one zastąpione daną *f* o wartości 0. U< jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

U* (*a b* — — *dc*)
Wykonanie operacji U* (*U-star*) powoduje zastąpienie — na stosie parametrów — danych słowowych bez znaku *a* i *b* ich iloczynem dwusłowowym bez znaku *dc*. U* jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

U. (*v* — —)
Wykonanie operacji U. (*U-dot*) powoduje wyprowadzenie danej słowowej bez znaku *v*, jako liczby całkowitej zakończonej jedną spacją. U. jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: U. 0 D. ;

U/ (*da b* — — *r q*)
Wykonanie operacji U/ (*U-slash*) powoduje zastąpienie — na stosie parametrów — danej dwusłowowej bez znaku *da* oraz danej słowowej bez znaku *b* parą danych *r* i *q*, tak dobranych, że *q* = *da*/*b*, zaś *r* jest resztą z dzielenia *da* przez *b*. U/ jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

UNTIL (*a n*1 — —) por. opis operacji BEGIN
Wykonanie operacji UNTIL (*until*) zaczyna się od upewnienia, że dana *n*1 ma wartość 1. Jeśli tak nie jest, to operacja UNTIL jest użyta w niepoprawnym kontekście. Spowoduje to wykonanie czynności przedstawionych w opisie operatora ?PAIRS. W przeciwnym razie w słowniku zostanie umieszczone wskazanie operatora 0BRANCH, a za nim adres względny tego operatora, który został umieszczony w słowniku pod adresem *a*. UNTIL jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: UNTIL 1 ?PAIRS COMPILE 0BRANCH
BACK ; IMMEDIATE

UPDATE (— —)
Wykonanie operacji UPDATE (*update*) powoduje ustawienie bitu *update* w pierwszym bicie ostatnio użytego bufora dyskowego w pamięci operacyjnej. UPDATE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)
: UPDATE PREV @ @ 8000 OR PREV @ ! ;

USE (— — *a*)
Wykonanie operacji USE (*use*) powoduje umieszczenie — na stosie parametrów — adresu zmiennej słowowej określającej adres tego bufora, który zostanie użyty jako następny. USE jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

USER (*a* --)

Wykonanie operacji USER (*user*) powoduje utworzenie definicji operatora o nazwie identycznej z nazwą najbliższego słowa występującego w strumieniu wejściowym. W polu parametrów tak utworzonej definicji zostanie umieszczona dana słowowa *a*. Dana ta stanowi adres względny liczony względem początku obszaru użytkownika (*user area*). Wykonanie operacji określonej przez tak zdefiniowany operator spowoduje umieszczenie — na stosie parametrów — adresu rzeczywistego bajtu o adresie względnym *a*. USER jest operatorem biernym, zdefiniowanym za pomocą operatora : (dwukropek)

: USER CONSTANT ;CODE *kod-maszynowy*

(*kod-maszynowy* reprezentuje program w kodzie maszynowym, którego adres zostanie umieszczony w polu kodu każdego operatora zdefiniowanego za pomocą kompilatora USER. Kod ten jest tak dobrany, aby jego wykonanie spowodowało umieszczenie — na stosie parametrów — sumy danej słowowej znajdującej się w polu parametrów wspomnianego operatora i adresu początku obszaru użytkownika.)

VARIABLE (*n* --)

Wykonanie operacji VARIABLE (*variable*) powoduje utworzenie definicji operatora o nazwie identycznej z nazwą najbliższego słowa występującego w strumieniu wejściowym. W polu parametrów tak utworzonej definicji zostanie umieszczona dana słowowa *n*. Dana ta stanowi wartość początkową zmiennej identyfikowanej przez właśnie utworzony operator. Wykonanie operacji określonej przez taki operator spowoduje umieszczenie — na stosie parametrów — adresu wspomnianej zmiennej. VARIABLE jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

: VARIABLE CONSTANT ;CODE *kod-maszynowy*

(*kod-maszynowy* reprezentuje program w kodzie maszynowym, którego adres zostanie umieszczony w polu kodu każdego operatora zdefiniowanego za pomocą kompilatora VARIABLE. Kod ten jest tak dobrany, aby jego wykonanie spowodowało umieszczenie — na stosie parametrów — adresu zmiennej słowowej, od której zaczyna się pole prametrów wspomnianego operatora.)

VLIST (--)

Wykonanie operacji VLIST (*V-list*) powoduje wyprowadzenie nazw operatorów zdefiniowanych w bieżącym podsłowniku wyszukiwania operatorów oraz w tych wszystkich podsłownikach, którym dany słownik jest bezpośrednio lub pośrednio podrzędny. W szczególności wykonanie omawianej operacji spowoduje wyprowadzenie nazw wszy-

stkich operatorów zdefiniowanych w podsłowniku FORTH. Jeśli podczas wyprowadzania nazw zostanie wprowadzony znak z klawiatury, to spowoduje to zakończenie wykonywania operacji VLIST. VLIST jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: VLIST 80 OUT ! CONTEXT @ @
        BEGIN
          OUT @ C/L >
          IF CR 0 OUT ! THEN
          DUP ID. SPACE PFA LFA @
          DUP 0= ?TERMINAL OR
        UNTIL
        DROP ;
```

VOC-LINK (-- *a*)

Wykonanie operacji VOC-LINK (*voke-link*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej słowowej, określającej adres pola kodu w definicji pseudooperatora, występującej w polu parametrów operatora identyfikującego ostatnio utworzony podsłownik. VOC-LINK jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

VOCABULARY (--)

Wykonanie operacji VOCABULARY (*vocabulary*) powoduje utworzenie definicji operatora identyfikującego podsłownik. Nazwa tego operatora jest identyczna z nazwą najbliższego słowa występującego w strumieniu wejściowym. W polu parametrów tak skompilowanej definicji zostanie umieszczone wskazanie pierwszego bajtu ciągu wskazań operatorów 2+, CONTEXT i !, a za nim definicja pseudooperatora, którego nazwą jest spacja. Wykonanie operacji określonej przez tak zdefiniowany operator spowoduje przypisanie zmiennej identyfikowanej przez operator CONTEXT adresu pola łącznika tego pseudooperatora, który jest zawarty w definicji omawianego operatora. VOCABULARY jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: VOCABULARY <BUILDS
                 A081 , CURRENT @ CFA ,
                 HERE VOC—LINK @ , VOC—LINK !
               DOES>
                 2+ CONTEXT ! ; IMMEDIATE
```

WARNING (-- *a*)

Wykonanie operacji WARNING (*warning*) powoduje umieszczenie — na stosie parametrów — adresu *a* zmiennej słowowej, której

wartość określa sposób reagowania interpretera na błędy. Jeśli zaistnieje błąd, a wspomniana zmienna ma wartość ujemną, to nastąpi wstrzymanie interpretacji i wykonanie operacji (ABORT). Jeśli zmienna ma wartość nieujemną, to zostanie wyprowadzony komunikat, a następnie zostanie wykonana operacja QUIT. Jeśli zmienna ma wartość 0, to komunikat zostanie uproszczony do podania numeru błędu. WARNING jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

WHILE (*a n*1 -- *a* 1 *b* 4) por. opis operacji BEGIN

Wykonanie operacji WHILE (*while*) zaczyna się od upewnienia, że dana *n*1 ma wartość 1. Jeśli tak nie jest, to operacja WHILE jest użyta w niepoprawnym kontekście. Spowoduje to wykonanie czynności przedstawionych w opisie operatora ?PAIRS. W przeciwnym razie w słowniku zostanie umieszczone wskazanie operatora 0BRANCH, a za nim dana słowowa o wartości 0. Ponadto, na stosie parametrów zostanie umieszczony adres *b* wspomnianej danej słowowej i dana o wartości 4. WHILE jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: WHILE [COMPILE] IF 2+ ; IMMEDIATE
```

WIDTH (-- *a*)

Wykonanie operacji WIDTH (*width*) powoduje umieszczenie — na stosie parametrów — adresu zmiennej słowowej określającej maksymalną liczbę zapamiętanych w słowniku znaków nazwy operatora. Liczba ta nie może przekraczać 31. WIDTH jest operatorem biernym, zdefiniowanym za pomocą kompilatora USER.

WORD (*c* --)

Wykonanie operacji WORD (*word*) zaczyna się od zdefiniowania dolnego bajtu danej słowowej *c* jako ogranicznika. Następnie są pomijane znaki ogranicznika występujące w strumieniu wejściowym, a znaki następne, poprzedzone jednobajtowym licznikiem znaków, są umieszczane w polu rozpoczynającym się od pierwszego wolnego bajtu słownika. Wprowadzanie kończy się po napotkaniu ogranicznika, a za ostatnim wprowadzonym znakiem zostaje umieszczona przynajmniej jedna spacja. Zakończenie wprowadzania znaków ma miejsce także wtedy, gdy w strumieniu wejściowym zostanie napotkany znak cr albo znak o kodzie 0. WORD jest operatorem biernym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: WORD BLK @
       IF
         BLK @ BLOCK
```

```
ELSE
  TIB @
THEN
IN @ + SWAP ENCLOSE HERE 22 BLANKS
IN +! OVER - >R R HERE C! + HERE
1+ R> CMOVE ;
```

XOR (*a b* -- *c*)

Wykonanie operacji XOR (X-O-R) powoduje zastąpienie — na stosie parametrów — danych słowowych *a* i *b* daną słowową *c* tak dobraną, że stanowi ona rezultat różnicy symetrycznej wyznaczonej równolegle na wszystkich parach odpowiadających sobie bitów danych *a* i *b*. XOR jest operatorem biernym, zdefiniowanym w kodzie maszynowym.

[(--)

Wykonanie operacji [(*left-bracket*) powoduje wprowadzenie interpretera w stan wykonywania. [jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: [ 0 STATE ! ; IMMEDIATE
```

[COMPILE] (--)

Wykonanie operacji [COMPILE] powoduje umieszczenie w słowniku wskazania operatora o nazwie identycznej z nazwą najbliższego słowa występującego w strumieniu wejściowym. Po wykonaniu tej czynności wspomniane słowo strumienia wejściowego jest pomijane. [COMPILE] jest operatorem czynnym zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: [COMPILE] -FIND 0= 0 ?ERROR DROP
            CFA , ; IMMEDIATE
```

] (--)

Wykonanie operacji] (*right-bracket*) powoduje wprowadzenie interpretera w stan kompilacji.] jest operatorem czynnym, zdefiniowanym za pomocą kompilatora : (dwukropek)

```
: ] C0 STATE ! ; IMMEDIATE
```

Dodatek A

Znaki kodu ASCII

00 nul	20 *spacja*	40 @	60 ‘
01 soh	21 !	41 A	61 a
02 stx	22 ”	42 B	62 b
03 etx	23 #	43 C	63 c
04 cot	24 $	44 D	64 d
05 enq	25 %	45 E	65 e
06 ack	26 &	46 F	66 f
07 bel	27 ’	47 G	67 g
08 bs	28 (	48 H	68 h
09 ht	29)	49 I	69 i
0A lf	2A *	4A J	6A j
0B vt	28 +	4B K	68 k
0C ff	2C ,	4C L	6C l
0D cr	2D –	4D M	6D m
0E so	2E .	4E N	6E n
0F si	2F /	4F O	6F o
10 dle	30 0	60 P	70 p
11 dc1	31 1	51 Q	71 q
12 dc2	32 2	52 R	72 r
13 dc3	33 3	53 S	73 s
14 dc4	34 4	54 T	74 t
15 nak	35 5	55 U	75 u
16 syn	36 6	56 V	76 v
17 etb	37 7	57 W	77 w
18 can	38 8	58 X	78 x
19 em	39 9	59 Y	79 y
1A sub	3A :	5A Z	7A z
1B esc	3B ;	5B [	7B {
1C fs	3C <	5C \	7C |
1D gs	3D =	5D]	7D }
1E rs	3E >	5E ^	7E ~
1F us	3F ?	5F _	7F del

Dodatek B

Rozkazy mikroprocesora Intel 8080

W mikroprocesorze Intel 8080 można wyodrębnić zespół 8-bitowych rejestrów oznaczanych literami A, F, B, C, D, E, H, L oraz dwa rejestry 16-bitowe: SP i PC.

Rejestr A jest akumulatorem, F — rejestrem znaczników, PC — licznikiem rozkazów, a SP — wskaźnikiem stosu. Z punktu widzenia pewnych rozkazów pary rejestrów: BC, DE i HL są traktowane jak rejestry dwubajtowe. W każdej z takich par drugi z elementów stanowi część mniej znaczącą. W podobny sposób można „tworzyć" dwubajtowy rejestr PSW z rejestrów A-F.

W rejestrze F występują bity, które mogą otrzymywać wartości 0 albo 1

Z = 1 oznacza, że rezultatem operacji było zero;
S = 1 oznacza, że najbardziej znaczący bit rezultatu miał wartość 1;
P = 1 oznacza, że suma modulo 2 bitów rezultatu jest równa zeru;
C = 1 oznacza, że w następstwie wykonania operacji powstało przeniesienie z najbardziej znaczącej pozycji;
AC = 1 oznacza, że powstało przeniesienie połówkowe, tj. przeniesienie z najbardziej znaczącego bitu dolnej tetrady.

Przykład

W celu zilustrowania zasad ustawiania bitów rejestru F, zostanie rozpatrzone wykonanie rozkazu dodawania i odejmowania argumentu bezpośredniego zero od wyzerowanego akumulatora, tj. wykonanie par rozkazów

```
XRA A          XRA A
       oraz
ADI 0          SUI 0
```

Jak wynika z dalszego opisu, rozkaz XRA służy do równoległego obliczania różnicy symetrycznej, a więc rezultatem XRA A jest ciąg 8 bitów

zero, co powoduje następujące ustawienie bitów rejestru F:

$Z = 1 \quad S = 0 \quad P = 1 \quad C = 0 \quad AC = 0$

Wykonanie w tej sytuacji rozkazu ADI powoduje zsumowanie dwóch bajtów zer i ponowne ustawienie rejestru F na przytoczone uprzednio wartości.

Inaczej rzecz ma się z rozkazem SUI. Ponieważ odejmowanie w zapisie uzupełnieniowym do 2 jest realizowane jako złożenie operacji dodania do odjemnej zanegowanego odjemnika i wartości 1, przebieg operacji jest następujący:

	00000000	odjemna
−		
	00000000	odjemnik
	00000000	odjemna
+	11111111	zanegowany odjemnik
+	1	wartość 1
	100000000	wynik sumowania
	00000000	rezultat

Ponieważ rezultat składa się z samych zer, nastąpi ustawienie $Z = 1$, $S = 0$, $P = 1$.

Ponieważ powstało przeniesienie z najbardziej znaczącej pozycji rezultatu wydawałoby się, że nastąpi ustawienie bitu $C = 1$. W istocie, ponieważ w mikroprocesorze Intel 8080 przyjęto, że wykonanie odejmowania powoduje *zanegowanie* bitu C, nastąpi ustawienie $C = 0$. Taki sposób potraktowania przeniesienia nie dotyczy natomiast przeniesienia połówkowego, skutkiem czego nastąpi ustawienie bitu $AC = 1$. Ostatecznie, wykonanie rozkazu SUI spowoduje w omawianym przypadku następujące ustawienie bitów rejestru F:

$Z = 1 \quad S = 0 \quad P = 1 \quad C = 0 \quad AC = 1$ □

W przytoczonych dalej opisach rozkazów użyto zwrotu „aktualizowany jest rejestr F" dla zwrócenia uwagi, że w następstwie wykonania odpowiedniego rozkazu i jego wariantów zostaną ustawione bity Z, S, P, C i AC rejestru znaczników. W pozostałych przypadkach bity rejestru F albo nie ulegają zmianie, albo zmieniają się tylko niektóre z nich — te, które jawnie wymieniono.

Ze względu na to, że przytoczony opis listy rozkazów ma charakter referencyjny, zastosowano nieformalną notację, której pewne elementy wymagają wyjaśnienia

- symbolem „ := " oznaczono przypisanie, a symbolem „ :=: " wymianę zawartości;
- małą literą *d* albo *s* oznaczono dowolny z rejestrów A, B, C, D, H, L, a małą literą *a* oznaczono zmienną znajdującą się w pamięci pod adresem *a* (np. w rozkazie LHLD) albo adres *a* (np. w rozkazie JMP);

• parą małych liter *dd* albo *ss* oznaczono zespół sprzężonych rejestrów (np. PSW, HL) – w każdym takim przypadku podając po dwukropku dopuszczalne wartości, podstawiane w miejsce wspomnianych liter;

• małą literą *i* oznaczono wartość bezpośrednią – jednobajtową, a parą liter *ii* oznaczono wartość bezpośrednią – dwubajtową, chyba że jawnie podano inne wymaganie;

• identyfikatorem *carry* oznaczono zmienną o wartości równej bitowi C rejestru F, a identyfikatorem *aux.carry* analogiczną zmienną związaną z bitem AC;

• symbolem „@" oznaczono adresowanie pośrednie (np. HL:=:@SP opisuje wymianę zawartości rejestru sprzężonego HL i dwubajtowej zmiennej spod adresu zawartego w SP);

• symbolem *carry.A* oznaczono złączenie bitu C rejestru F i bitów akumulatora, od najbardziej znaczącego – A7, do najmniej znaczącego – A0;

• symbolami *rotl* i *rotr* oznaczono operację obrotu – odpowiednio w lewo i w prawo.

Komentarz należy się także operacjom *push* i *pop*. Pierwsza z nich powoduje odesłanie na stos pary sprzężonych rejestrów – najpierw bardziej znaczącego, a następnie mniej znaczącego. Druga wykonuje czynność odwrotną – zdejmuje parę bajtów ze stosu, umieszczając pierwszy w mniej znaczącej, a drugi w bardziej znaczącej części rejestru sprzężonego.

Rozkazy przesyłania

MOV – przenieść zawartość

MOV *d,s* ; *d*:=*s*
MOV M,*s* ; @HL:=*s*
MOV *d*,M ; *d*:=@HL

• rejestr F nie ulega zmianie.

MVI – wstaw wartość

MVI *d,i* ; *d*:=*i*
MVI M,*i* ; @HL:=*i*

• rejestr F nie ulega zmianie.

LDA – załaduj bezpośrednio akumulator

LDA *a* ; A:=*a*

• rejestr F nie ulega zmianie

LDAX – załaduj pośrednio akumulator

LDAX *ss* ; A:=@*ss* *ss:* B,D

• rejestr F nie ulega zmianie.

LXI — wstaw wartość
LXI *dd,ii*; *dd*:=*ii*
• rejestr F nie ulega zmianie.

LHLD — załaduj bezpośrednio HL
LHLD *a* ; HL:=*a*
• rejestr F nie ulega zmianie.

STA — zapamiętaj akumulator
STA *a* ; *a*:=A
• rejestr F nie ulega zmianie.

STAX — zapamiętaj pośrednio akumulator
STAX *dd* ; @*dd*:=A *dd*: B,D
• rejestr F nie ulega zmianie.

SHLD — zapamiętaj HL bezpośrednio
SHLD *a* ; *a*:=HL
• rejestr F nie ulega zmianie.

XCHG — wymień zawartość rejestrów HL i DE
XCHG ; HL:=:DE
• rejestr F nie ulega zmianie.

Arytmetyka

ADD — dodaj do akumulatora
ADD *s* ; A: = A+*s*
ADD M ; A:=A+@HL
• aktualizowany jest rejestr F.

ADI — dodaj do akumulatora wartość stałą
ADI *i* ; A: = A+*i*
• aktualizowany jest rejestr F.

ADC — dodaj do akumulatora łącznie z *carry*
ADC *s* ; A: = A+*s*+*carry*
ADC M ; A: = A+@HL+*carry*
• aktualizowany jest rejestr F.

ACI — dodaj do akumulatora stałą łącznie z *carry*
ACI *i* ; A: = A+*i*+*carry*
• aktualizowany jest rejestr F.

DAD – dodaj do HL

DAD *ss* ; HL: = HL+*ss* *ss*: B,D,H,SP

• w rejestrze F aktualizowany jest tylko bit C.

SUB – odejmij od akumulatora

SUB *s* ; A: = A–*s*

SUB M ; A: = A–@HL

• aktualizowany jest rejestr F; stan bitu C odzwierciedla orzeczenie: „odjemna mniejsza od odjemnika" (dla prawdy C = 1, dla fałszu C = 0).

SUI – odejmij od akumulatora stałą

SUI *i* ; A: = A–*i*

• aktualizowany jest rejestr F (z uwagą jak dla rozkazu SUB).

SBB – odejmij od akumulatora łącznie z *carry*

SBB *s* ; A: = A–*s*–*carry*

SBB M ; A: = A–@HL–*carry*

• aktualizowany jest rejestr F (z uwagą jak dla rozkazu SUB).

SBI – odejmij od akumulatora stałą łącznie z *carry*

SBI *i* ; A: = A–*i*–*carry*

• aktualizowany jest rejestr F (z uwagą jak dla rozkazu SUB).

Inkrementacja, dekrementacja

INR – inkrementacja bajtowa

INR *d* ; *d*: = *d*+1

INR M ; @HL: = @HL+1

• aktualizowany jest rejestr F, z wyjątkiem bitu C, który nie ulega zmianie.

INX – inkrementacja dwubajtowa

INX *dd*; *dd*; = *dd*+1 *dd*: B,D,H,SP

• rejestr F nie ulega zmianie.

DCR – dekrementacja bajtowa

DCR *d* ; *d*: = *d*–1

DCR M ; @HL: = @HL–1

• aktualizowany jest rejestr F, z wyjątkiem bitu C, który nie ulega zmianie.

DCX — dekrementacja dwubajtowa

DCX *dd* ; *dd*: = *dd*−1 *dd*: B,D,H,SP

• rejestr F nie ulega zmianie.

Rozkazy sterujące

Do kategorii rozkazów sterujących zostały zaliczone przejścia, wywołania, powroty z podprogramów i przerwań i zmiany rejestru PC. Dla skrócenia opisu, symbolem *f* oznaczono kryterium wykonania przejścia, wywołania albo powrotu. Kryterium takim może być

Z — *zero* — jedynkowy bit Z w rejestrze F,
NZ — *not zero* — zerowy bit Z w rejestrze F,
C — *carry* — jedynkowy bit C w rejestrze F,
NC — *not carry* — zerowy bit C w rejestrze F,
PR — *parity even* — jedynkowy bit P w rejestrze F,
PO — *parity odd* — zerowy bit P w rejestrze F,
M — *minus* — jedynkowy bit S w rejestrze F,
P — *plus or zero* — zerowy bit S w rejestrze F.

JMP, J*f* — rozkazy przejścia

JMP *a* ; PC: = *a*

J*f* *a* ; **if** *f* **then** PC: = *a*

• rejestr F nie ulega zmianie.

• przykładem rozkazu J*f* jest

JNZ 5 ; jeśli rejestr F zawiera zerowy bit Z, wykonaj przypisanie PC: = 5.

CALL, C*f* — rozkaz wywołania

CALL *a* ; *push* PC, PC: = *a*

C*f* *a* ; **if** *f* **then** CALL *a*

• rejestr F nie ulega zmianie.

RET, R*f* — rozkaz powrotu

RET ; *pop* PC

R*f* ; **if** *f* **then** RET

• rejestr F nie ulega zmianie.

RST — restart (obsłuż przerwanie)

RST *i* ; *push* PC, PC: = 8**i* ($i \leqslant 7$)

• rejestr F nie ulega zmianie.

PCHL — zmień rejestr PC

PCHL ; PC: = HL

• rejestr F nie ulega zmianie.

Rozkazy stosowe

PUSH — umieść na stosie
PUSH *ss* ; *push ss* *ss*; B,D,H,PSW
• rejestr F nie ulega zmianie.

POP — zdejmij ze stosu
POP *dd*; *pop dd* *dd*: B,D,H,PSW
• rejestr F nie ulega zmianie, chyba że *dd* = PSW, kiedy to przybiera wartość pochodzącą ze stosu.

XTHL — wymień HL ze szczytem stosu
XTHL ; HL: = :@SP
• rejestr F nie ulega zmianie.

SPHL — zmień położenie stosu
SPHL ; SP: = HL
• rejestr F nie ulega zmianie.

Rozkazy logiczne

ANA — iloczyn logiczny akumulatora
ANA *d* ; A: = A **and** *d*
ANA M ; A: = A **and** @HL
• aktualizowany jest rejestr F z wyjątkiem bitów C i AC, które są zerowane.

ANI — iloczyn logiczny akumulatora ze stałą
ANI *i* ; A: = A **and** *i*
• aktualizowany jest rejestr F, z wyjątkiem bitów C i AC, które są zerowane.

ORA — suma logiczna akumulatora
ORA *d* ; A: = A **or** *d*
ORA M ; A: = A **or** @HL
• aktualizowany jest rejestr F z wyjątkiem bitów C i AC, które są zerowane.

ORI — suma logiczna akumulatora i stałej
ORI *i* ; A: = A **or** *i*
• aktualizowany jest rejestr F, z wyjątkiem bitów C i AC, które są zerowane.

XRA — różnica symetryczna akumulatora

XRA *d* ; A: = A **xor** *d*

XRA M ; A: = A **xor** @HL

• aktualizowany jest rejestr F, z wyjątkiem bitów C i AC, które są zerowane.

XRI — różnica symetryczna akumulatora i stałej

XRI *i* ; A: = A **xor** *i*

• aktualizowany jest rejestr F, z wyjątkiem bitów C i AC, które są zerowane.

CMA — negacja akumulatora

CMA ; A: = **not** A

• rejestr F nie ulega zmianie.

Porównania

CMP — porównanie akumulatora

CMP *ss* ; ? A—*ss*

CMP M; ? A—@HL

• aktualizowany jest rejestr F; w szczególności jeśli A<@HL, to bit C otrzymuje wartość 1, a gdy A = @HL to Z otrzymuje wartość 1.

CPI — porównanie akumulatora ze stałą

CPI *i* ; ? A—*i*

• aktualizowany jest rejestr F (por. CMP).

Przesunięcia

Wszystkie przesunięcia dotyczą tylko akumulatora i są jednokrotne.

RLC — obrót w lewo z ustawieniem *carry*

RLC *carry*: = A7, *rotl* A

• w rejestrze F zmianie ulega tylko bit C, który przyjmuje wartość bitu A7.

RRC — obrót w prawo z ustawieniem *carry*

RRC ; *carry*: = A0, *rotr* A

• w rejestrze F zmianie ulega tylko bit C, który przyjmuje wartość bitu A0.

RAL — obrót w lewo wraz z *carry*

RAL ; *rotl carry*.A

• w rejestrze F zmianie ulega tylko bit C, który przyjmuje wartość bitu A7 (przed obrotem).

RAR — obrót w prawo wraz z *carry*

RAR ; *rotr carry*.A

• w rejestrze F zmianie ulega tylko bit C, który przyjmuje wartość bitu A0 (przed obrotem).

Wejście/wyjście

IN — wprowadzenie z portu

IN *i* ; A: = port(*i*)

• rejestr F nie ulega zmianie.

OUT — wyprowadzenie do portu

OUT *i* ; port(*i*): = A

• rejestr F nie ulega zmianie.

Rozkazy różne

CMC — negacja *carry*

CMC ; *carry*: = *not carry*

• w rejestrze F zmianie ulega tylko bit C, który przyjmuje wartość zanegowaną.

STC — ustawienie *carry*

STC ; *carry*: = 1

• w rejestrze F zmianie ulega tylko bit C, który przyjmuje wartość 1.

EI — odblokowanie przerwań

EI ; *enable interrupt*

• rejestr F nie ulega zmianie.

DI — zablokowanie przerwań

DI ; *disable interrupt*

• rejestr F nie ulega zmianie.

HLT — czekaj na przerwanie

HLT ; *wait*

• rejestr F nie ulega zmianie.

NOP — nic nie rób

NOP ;

• rejestr F nie ulega zmianie.

DAA – dodaj poprawkę

DAA ; **if** *aux.carry* **or** A(3..0) >9 **then**
A: = A+6
if *carry* **or** A(7..4)>9 **then**
A: = A+16*6

● aktualizowany jest rejestr F.

Literatura

1. Bielecki J.: fig-Forth, definiowanie kompilatorów i instrukcji strukturalnych. *Informatyka*, 1985, nr 7.
2. Bielecki J.: Programowanie asemblerowe w języku fig-Forth. *Informatyka*, 1985, nr 11.
3. Bielecki J., Prochowski G., Szeffer J.: Oprogramowanie narzędziowe do programowania w języku Forth. PW, Instytut Informatyki, 1984.
4. Brodie L.: *Starting Forth*. Englewood Cliffs, New Jersey, Prentice-Hall 1981.
5. Derrick M.: *Forth Encyclopedia*. Central Book Co. 1982.
6. McCabe C. K.: *Forth Fundamentals*. Beaverton, Dilithium Press 1983.
7. Thomasson D.: *Advanced Spectrum Forth*. London, Melbourne House 1984.

Skorowidz

Skorowidz operatorów

WYDAWNICTWA NAUKOWO-TECHNICZNE
ul. Mazowiecka 2/4, 00-048 Warszawa
tel. 26-72-71 do 79
Dział Upowszechniania i Sprzedaży
tel. 27-56-87

WNT Warszawa 1988.
Wydanie I. Nakład 9 700+300 egz.
Ark. wyd. 9,6. Ark. druk. 10,25.
Format B5. Papier offset. kl. III 70 g.
Podpisano do druku w październiku 1988.
Druk ukończono w listopadzie 1988.
Symbol Et/82193/WNT.
Zam. 828/11.10/88/II U-61.
Szczecińskie Zakłady Graficzne.

Cena zł 580,-

ISBN 83-204-0930-6